



OPENcontrol

WinPLUS Library - User manual

DOCUMENT NUMBER: 45006947Y
EDITION: 05
REF. RELEASE: V3.2
AUTHOR: PRIMA ELECTRO S.p.A.
DATE: November 2013

PRIMA ELECTRO reserves the right to modify and improve the product described in this manual at any time and without prior notice. The application of this manual is under customer responsibility. No further guarantees will be given by PRIMA ELECTRO, in particular for any possible faults, incompleteness and/or difficulties in the operation. In no event will PRIMA ELECTRO responsible or liable for indirect or consequential damages that may result by the use of such documentation.

COPYRIGHT 2013 - PRIMA ELECTRO S.p.A.

All rights of reproduction are reserved. Reproduction, use or disclosure to third parties without express authority is *strictly forbidden*.

Edition	Date	Note	Author
05	22/11/13		Prima Electro S.p.A.

Contents

1. Preface	17
1.1 Introduction	17
1.2 Reading keys & security instructions	17
1.3 Functions reference list in alphabetical order	18
2. OPEN_CNC LIBRARY: Functions for communication with CNC processes.....	29
2.1 AxisShareON.....	32
2.2 AxisShareOFF.....	33
2.3 CycleStart.....	34
2.4 CycleStop.....	35
2.5 ExecMDIBlock	36
2.6 ExecBlock.....	37
2.7 HandWheelON.....	38
2.8 HandWheelExe	39
2.9 HandWheelOFF	41
2.10 HandWheelParam	42
2.11 FeedBypassON	44
2.12 FeedBypassOFF	45
2.13 ActiveReset	46
2.14 HoldOFF	47
2.15 HoldON	48
2.16 EnterFeedHold	49
2.17 ExitFeedHold	51
2.18 MasterSlaveDefine	52
2.19 MasterSlaveGO	59
2.20 MasterSlaveParam.....	60
2.21 MasterSlaveStatus	61
2.22 MasterSlaveSTOP	62
2.23 MasterSlaveUndefine.....	63
2.24 PP_BlockSearch.....	64
2.25 PP_BlockNumberSearch.....	65

2.26	PP_ExecUntilBlockNumber	66
2.27	PP_ExecUntil	67
2.28	PP_ExecUntilNext	68
2.29	PP_GetBlockNumber	70
2.30	PP_GetMarkers	71
2.31	PP_MarkerSearch	72
2.32	PP_GetName	73
2.33	PP_LineSearch	74
2.34	PP_Resume	75
2.35	PP_RunInfoGO	76
2.36	PP_RunInfoRead	77
2.37	PP_RunInfoStatus	79
2.38	PP_RunInfoSTOP	80
2.39	PP_Select	81
2.40	PP_TextSearch	83
2.41	PP_LoadXmlMap	84
2.42	ProtectAreaDefine	85
2.43	ProtectAreaDisable	87
2.44	ProtectAreaEnable2D	88
2.45	ProtectAreaEnable2D_2	90
2.46	ReadProcessInfo	92
2.47	SetProcessMode	97
2.48	Reset	98
2.49	EndReset	100
2.50	SelectAxis	101
2.51	SelectAxisAndOverride	102
2.52	SetAxisManualOverride	104
2.53	EndOfBlockdDisable	106
2.54	EndOfBlockEnable	107
2.55	SetFilterConsole	108
2.56	ReadFilterConsole	110
2.57	GetStrobeConsole	114
2.58	AckStrobeConsole	116
2.59	AxisGet	120
2.60	GetStrobePProgram	123

2.61	ClearStrobeConsole	125
2.62	SetFilterPProgram.....	127
2.63	ReadFilterPProgram	129
2.64	AckStrobePProgram	136
2.65	ClearStrobePProgram	138
2.66	GetJogIncrement.....	140
2.67	SetJogIncrement.....	141
2.68	SetManualOverride	142
2.69	SetMultiBlockRetrace	144
2.70	SetSearchInMemory	145
2.71	SetRapidOverride	146
2.72	SetFeedOverride.....	147
2.73	SetSpindleOverride	148
2.74	SpindleGet	149
2.75	SpindleRelease	150
2.76	TeachPendOFF	151
2.77	TeachPendON.....	152
2.78	TOOL_ActivOffset.....	153
2.79	TOOL_CalcOffset.....	155
2.80	AXIS_ActivOffset.....	158
2.81	AXIS_CalcOffset.....	160
2.82	TOOL_GetSlaveID	162
2.83	VAR_Read	163
2.84	VAR_ReadArray	165
2.85	VAR_Write	167
2.86	VAR_WriteProtect	169
2.87	VAR_WriteArray	171
2.88	VAR_ReadEx.....	173
2.89	VAR_WriteEx.....	175
2.90	AbortTappingCycle	177
2.91	SetDynamicAxOverride.....	178
2.92	ToolCenterPointCFGWrite	180
2.93	ToolCenterPointWrite	184
2.94	ToolCenterPointRestore	186
2.95	ToolCenterPointRead	188

2.96	ToolCenterPointCFGRead	190
2.97	ToolCenterPointON	194
2.98	ToolCenterPointOFF.....	195
2.99	ToolCenterPointStatus	196
2.100	SetDynamicOverride.....	198
2.101	Refresh3DPositions	199
2.102	CamAxiDefine.....	200
2.103	CamAxiStart.....	202
2.104	CamOutDefine	205
2.105	CamOutStart.....	207
2.106	CamStop.....	209
2.107	CamUndefine	210
	CNC error list (processes)	211
2.108	211

3. OPEN_MOTION LIBRARY: motion functions 223

3.1	GMC_InterpCreate	226
3.2	GMC_InterpDelete	228
3.3	GMC_InterpEnterFeedHold	229
3.4	GMC_InterpExitFeedHold	230
3.5	GMC_InterpEnterHold	231
3.6	GMC_InterpExitHold	232
3.7	GMC_EnterFeedHold	233
3.8	GMC_ExitFeedHold	234
3.9	GMC_EnterHold	235
3.10	GMC_ExitHold	236
3.11	GMC_InterpModify	237
3.12	GMC_InterpReadInfo	239
3.13	GMC_InterpReset	242
3.14	GMC_Reset	243
3.15	GMC_Move	244
3.16	GMC_MoveJog	248
	GMC_MoveOnPosition	250
3.17	250
3.18	GMC_MoveOnSignal	255
3.19	GMC_MoveUntilSignal	259

3.20	GMC_Poly.....	264
3.21	GMC_ProtectAreaCmd.....	268
3.22	GMC_MoveRoundOFF.....	269
3.23	GMC_MoveUntilVariable	273
3.24	GMC_ToleranceParams	278
3.25	GMC_CircleCCW	281
3.26	GMC_CircleCCWRadius.....	287
3.27	GMC_CircleCW	293
3.28	GMC_CreateAxVirtual.....	300
3.29	GMC_DeleteAxVirtual	302
3.30	GMC_CircleCWRadius	303
3.31	GMC_ContinuousAbort.....	309
3.32	GMC_ContinuousEnd.....	310
3.33	GMC_ContinuousParallel.....	311
3.34	GMC_ContinuousParam	315
3.35	GMC_ContinuousStart.....	316
3.36	GMC_HwndExe.....	319
3.37	GMC_HwndOFF	321
3.38	GMC_HwndON	322
3.39	GMC_HwndParam.....	323
3.40	GMC_HwndParamExt.....	324
3.41	GMC_MSLdefine.....	326
3.42	GMC_MSLexe	328
3.43	GMC_MSLexeONposition.....	330
3.44	GMC_MSLexeONSignal.....	333
3.45	GMC_MSLfollow	335
3.46	GMC_MSLfollowONposition.....	337
3.47	GMC_MSLfollowONSignal.....	340
3.48	GMC_MSLjog	342
3.49	GMC_MSLjogONposition.....	344
3.50	GMC_MSLjogONSignal.....	346
3.51	GMC_MSLparam	348
3.52	GMC_MSLreset	355
3.53	GMC_MSLresetONposition.....	357
3.54	GMC_MSLresetONSignal.....	360

3.55	GMC_MSLundef.....	362
3.56	GMC_MSLrealign.....	364
3.57	GMC_Probe.....	365
3.58	GMC_ProtectAreaDefine	370
3.59	GMC_ReadAxisInfo	372
3.60	GMC_ReadAxisInfoALL.....	378
3.61	GMC_ChangePosLoop	384
3.62	GMC_ChangePosLoopALL.....	386
3.63	GMC_SetAxisProperty	388
3.64	GMC_SetAxisPropertyALL.....	390
3.65	GMC_SetBitONposition	392
3.66	GMC_SetBitONvariable	394
3.67	GMC_SetFeedOverride.....	396
3.68	GMC_SetFeedOverrideALL.....	398
3.69	GMC_SetFeed	399
3.70	GMC_SetJerk.....	400
3.71	GMC_SetRamp	401
3.72	GMC_SetRampALL.....	405
3.73	GMC_SetOrigin.....	408
3.74	GMC_ActivateOrigin.....	410
3.75	GMC_SetResetQuote	412
3.76	GMC_ActivateResetQuote	414
3.77	GMC_SetResetQuoteONmarker	416
3.78	GMC_SetResetQuoteONPosition	417
3.79	GMC_SetResetQuoteONSignal.....	419
3.80	GMC_SetResetQuoteShift.....	421
3.81	GMC_Homing	423
3.82	GMC_SpindleChangeGear	425
3.83	GMC_SpindleG96.....	426
3.84	GMC_SpindleG97.....	428
3.85	GMC_SpindleOrient.....	430
3.86	GMC_WaitONposition	432
3.87	GMC_WaitONSignal	434
3.88	GMC_WaitONtimeout	435
3.89	GMC_WaitONvariable.....	436

3.90	GMC_Drill.....	437
3.91	GMC_Bore.....	440
3.92	GMC_Tapp.....	443
3.93	GMC_TappTransd	446
3.94	GMC_Thread	450
3.95	GMC_Circle3DCCW	454
3.96	GMC_Circle3D3P	460
3.97	GMC_Circle3DCW.....	464
3.98	GMC_SetDynamicOverride	470
3.99	GMC_SetDynamicOverrideALL.....	472
3.100	GMC_CamAxiDefine.....	474
3.101	GMC_CamAxiStart.....	477
3.102	GMC_CamAxiStartOnPosition.....	480
3.103	GMC_CamAxStartOnSignal	484
3.104	GMC_CamOutDefine	488
3.105	GMC_CamOutStart.....	490
3.106	GMC_CamOutStartOnPosition	492
3.107	GMC_CamOutStartOnSignal	495
3.108	GMC_CamStop.....	497
3.109	GMC_CamStopOnSignal	499
3.110	GMC_CamStopOnPosition.....	501
3.111	GMC_CamUndefine.....	503
3.112	Motion Control error list.....	504

4. OPEN_LIBRARY: General purpose functions..... 513

4.1	AsciiToInt.....	516
4.2	IntToAscii.....	517
4.3	Decoder	518
4.4	Encoder	519
4.5	InitData.....	520
4.6	SearchData	522
4.7	InitMemory.....	524
4.8	MoveData	526
4.9	MoveMemory.....	528
4.10	Delay	531
4.11	EnterCritical.....	532

4.12	ExitCritical	533
4.13	CriticalStatus	534
4.14	ReadInputsIntoWord	535
4.15	WriteOutputsFromWord	536
4.16	SendMessage	537
4.17	WarningMessage	538
4.18	SendSemaphore	539
4.19	WaitOnSemaphore	540
4.20	WaitOnSemaphoreStatus	541
4.21	EmergencyStopClose	542
4.22	EmergencyStopOpen	543
4.23	FastInputRead	544
4.24	FastOutputRead	545
4.25	FastOutputWrite	546
4.26	FileReadWrite	547
4.27	GetReleaseInfo	553
4.28	GetSystemSerialNumber	554
4.29	ME2_CommStatus	555
4.30	AckStrobeEmerg	556
4.31	ClearStrobeEmerg	558
4.32	GetStrobeEmerg	559
4.33	WinPlusEmergency	560
4.34	ReadFilterEmerg	561
4.35	WaitOnMessage	565
4.36	SetFilterEmerg	566
4.37	WarningMessageDisplay	568
4.38	TranslateError	570
4.39	TranslateErrorExt	571
4.40	GetBoardInfo	573
4.41	GetSystemInfo	577
4.42	OsWire_GetEmcyInfo	578
4.43	ProbeDelete	580
4.44	ProbeON	581
4.45	ProbeCreate	582
4.46	ProbeOFF	583

4.47	GetAxisIDFromName	584
4.48	GetAxisNameFromID	585
4.49	TBL_Lock.....	586
4.50	TBL_Unlock.....	587
4.51	TBL_ReadField	588
4.52	TBL_WriteField	591
4.53	TBL_ReadRecord	594
4.54	TBL_WriteRecord	596
4.55	TBL_SearchField	598
4.56	ConsoleOFF.....	601
4.57	ConsoleON.....	602
4.58	ConsoleReadInput.....	604
4.59	ConsoleReadOutput.....	606
4.60	ConsoleWriteOutput.....	608
4.61	ConsoleSetup.....	610
4.62	ConsoleSetupArray.....	613
4.63	Filter_Acc_Create.....	618
4.64	Filter_Acc_Run	619
4.65	Filter_Bessel_Delete.....	621
4.66	Filter_Boost_Create.....	622
4.67	Filter_Boost_Run	623
4.68	Filter_LowPass1st_Delete.....	625
4.69	Filter_Smooth_Create	626
4.70	Filter_Smooth_Run	627
4.71	Filter_Acc_Delete	629
4.72	Filter_Bessel_Create	630
4.73	Filter_Bessel_Run	631
4.74	Filter_Boost_Delete	633
4.75	Filter_LowPass1st_Create	634
4.76	Filter_LowPass1st_Run	635
4.77	Filter_Smooth_Delete	637
4.78	LookUpT_Linear_Delete	638
4.79	LookUpT_Spline3_Create	639
4.80	LookUpT_Spline3_Run	641
4.81	LookUpT_Linear_Create	643

4.82	LookUpT_Linear_Run.....	645
4.83	LookUpT_Spline3_Delete.....	647
4.84	Pid_Delete.....	648
4.85	Pid_Create	649
4.86	Pid_Run	650
5.	Open_EtherCAT library: EtherCAT functions	651
5.1	ECAT_CommStatus.....	652
5.2	ECAT_ReadDiagLog	654
5.3	ECAT_WriteDeviceData	656
5.4	ECAT_WriteMasterInfo	657
5.5	ECAT_ReadDeviceData	659
5.6	ECAT_GetSlaveInfo	660
5.7	ECAT_SetSlaveInfo	664
5.8	SOE_ReadParameter	666
5.9	SOE_Command	669
5.10	SOE_WriteParameter	671
5.11	Service Channel internal errors table	673
6.	Open_SERCOSIII library: SERCOSIII functions	675
6.1	S3_Command	676
6.2	S3_ReadDeviceData.....	678
6.3	S3_ReadParameter	679
6.4	S3_WriteDeviceData	682
6.5	S3_CommStatus	683
6.6	S3_ReadDiagLog	684
6.7	S3_WriteParameter	688
6.8	Service Channel internal errors table	690
7.	OPEN_AXIS LIBRARY: Axes motion functions.....	692
7.1	AXD_ReadParameter.....	694
7.2	AXD_RTFeedbackStart	696
7.3	AXD_WriteParameter	698
7.4	AXD_Command.....	700
7.5	AXD_RTFeedbackRead	702
7.6	AXD_RTFeedbackStop	703
7.7	AXD_RTTargetStart	704

7.8	AXD_RTTTargetWrite	706
7.9	AXD_RTTTargetStop	707
7.10	AX_ReadParameter	708
7.11	AX_WriteParameter	718
7.12	AX_ChangeServoMode	728
7.13	AX_CheckHardwareOvertravel	729
7.14	AX_SetHardwareOvertravel	730
7.15	AX_Disable	731
7.16	AX_Enable	733
7.17	AX_Reset	735
7.18	AX_ReadStatus	736
7.19	AX_SetStatus	739
7.20	AX_ReadPendingCommands	741
7.21	AX_Filter	743
7.22	AX_EnableProbe	745
7.23	AX_ReadProbePosition	746
7.24	AX_ReadProbeStatus	748
7.25	AX_SetTestMode	749
7.26	AX_ResetPosition	750
7.27	AX_SelectGearFoller	752
7.28	AX_SpindleSwitch	754
7.29	AX_CalibDisable	755
7.30	AX_CalibEnable	756
7.31	AX_CalibLoadFile	757
7.32	AX_CalibDisableGet	758
7.33	AX_CalibEnableGet	759
7.34	AX_CalibStatus	760
7.35	AX_ServoToPLC	761
7.36	AX_PLCToServo	763
7.37	RS_CreateResource	765
7.38	RS_DestroyResource	768
7.39	RS_SensorExecCmd	770
7.40	RS_SensorReadPar	773
7.41	RS_SensorWritePar	775
7.42	RS_SensorReadTable	777

7.43	RS_SensorWriteTable.....	778
7.44	RS_TransducerRead.....	780
7.45	RS_AnalogRead.....	781
7.46	RS_AnalogWrite.....	782
7.47	List of errors returned by Servo environment.....	783
7.48	Resources mapping within the Servo loop ambient	789
8.	4C_CANopen LIBRARY: CANopen FUNCTIONS	792
8.1	CANopen_BoardCmd	793
8.2	CANopen_GetEmcyInfo.....	795
8.3	CANopen_GetEmcyNode.....	798
8.4	CANopen_GetStatus	799
8.5	CANopen_NmtCmd.....	802
8.6	CANopen_ReadSdoCmd	804
8.7	CANopen_SyncCmd	807
8.8	CANopen_WriteSdoCmd	808
9.	Functions for the management of XML files - 4C_XMLFILE.....	811
9.1	XMLClose.....	812
9.2	XMLOpen	813
9.3	XMLRead	815
9.4	XMLWrite	817
9.5	XMLReadAttribute.....	819
9.6	XMLReadAttributeEx	822
9.7	XMLReadEx.....	825
9.8	XMLWriteAttribute.....	828
9.9	XMLRemove	831
9.10	XMLRemoveAttribute.....	833
9.11	XMLSave	835
10.	Generic system functions - 4C.Utility2.....	836
10.1	LoadLibrary	837
10.2	FreeLibrary	839
10.3	RunProgram	840
10.4	GetDate	841
10.5	GetTime	842
10.6	SetDate.....	843

10.7	SetTime	844
10.8	CloseDB.....	845
10.9	CpuUsage.....	846
10.10	DeleteDB	847
10.11	ForceRemapInput.....	848
10.12	OpenDB	852
10.13	ReadDB.....	854
10.14	ReadRemapOutputCFG	856
10.15	AddRemapOutputCFG.....	860
10.16	RemapOutput	864
10.17	RemapOutputByClass	865
10.18	DeleteRemapOutput.....	866
10.19	ResizeDB	867
10.20	CopyDB.....	869
10.21	CreateDB.....	871
10.22	FileListDB.....	873
10.23	ForceRemapOutput.....	875
10.24	PurgeRetainDB	879
10.25	ReadRemapInputCFG	880
10.26	AddRemapInputCFG.....	884
10.27	RemapInput.....	888
10.28	RemapInputByClass	889
10.29	DeleteRemapInput.....	890
10.30	GetRemapAddress.....	891
10.31	RenameDB	893
10.32	WriteDB.....	895
II.	Communication using TCP-IP and UDP Socket	897
11.1	SocketAccept.....	898
11.2	SocketClose	900
11.3	SocketCreate	901
11.4	SocketListen.....	903
11.5	SocketSelect.....	905
11.6	SocketBind.....	907
11.7	SocketConnect.....	909
11.8	SocketEnableNonblocking	911

11.9	SocketReceive.....	912
11.10	SocketSend	914
12.	<i>Serial line related functions – 4C_SERIALCOMM.....</i>	916
12.1	SerialLineClose	917
12.2	SerialLineOpen	918
12.3	SerialLineReadSetup.....	919
12.4	SerialLineWriteSetup.....	922
12.5	SerialLineReceive	925
12.6	SerialLineTransmit.....	927
12.7	SerialLineReset	929
12.8	SerialLineStatus.....	930
	List of errors returned by Serial Line	932

1. Preface

1.1 Introduction

This manual presents a description of all functions needed for the project of the OPENcontrol numerical controls machine logic applications.

Each function has a Structured Text (ST) code and the input/output variables description.

In particular, this manual analyses functions and function blocks. For additional information on WinPLUS language and its application with CNC OPENcontrol, please refer to the following manuals:

OPENcontrol - OPEN10 Application Manual	45006942N
OPENcontrol - OPEN20/30 Application Manual	45006962W
4Control - Programming Manual	45004612A

Functions can be executed using different synchronization methods (WAIT, NO-WAIT etc.). For an explanation of all synchronisms, please refer to “OPENcontrol - OPEN10 Application manual” or “OPENcontrol - OPEN20/30 Application manual”.

Further on in the manual, the “Functions reference list in alphabetical order” can help in finding a specific function or FB.

The list presents the functions in alphabetical order providing a short description together with the page number where the function is presented in the manual.

In the manual, the function blocks are divided by functional groups instead.

1.2 Reading keys & security instructions

ATTENTION! To use the system in a correct manner, follow the instructions provided in this manual and pay special attention to the indications below.



WARNING! Dangerous Voltage: This symbol is associated with high dangerous voltage that could damage the system, the equipment and the operators.



WARNING! Danger: This symbol is associated with facts and circumstances that could damage the system, the equipment and the operators.



NOTES: This symbol is used for operations that have to be executed with great care in order to ensure their successful completion.

1.3 Functions reference list in alphabetical order

AbortTappingCycle	Aborts the tapping cycle	178
AckStrobeConsole	Informs the system that a console event has been acquired by the PLC	117
AckStrobeEmerg	Informs the system that an emergency has been acquire by the PLC	557
AckStrobePProgram	Informs the system that a partprogram event has been acquired by the PLC	137
ActiveReset	Requests the active reset function forform the process	47
AddRemapInputCFG	Allows to upload in memory a file of inputs remap definitions, without deleting the definition already uploaded	885
AddRemapOutputCFG	Allows to upload in memory a file of output remap definitions, without deleting the definitions previously uploaded.	860
AsciiToInt	Converts a character into its corresponding ASCII numerical value	516
AX_CalibDisable	Disables all the axis generated offsets	755
AX_CalibDisableGet	Disables for one axis the compensations due to another axis	758
AX_CalibEnable	Enables an axis in receiving all available offsets	756
AX_CalibEnableGet	Enables for one axis the compensations due to another axis	759
AX_CalibLoadFile	Uploads the axis offset file	757
AX_CalibStatus	Returns Word status offsets enabled on an axis	760
AX_ChangeServoMode	Changes servo loop mode	728
AX_CheckHardwareOvertravel	Manages the hardware limit switches of an axis	729
AX_Disable	Disables the axis servo loop modes	731
AX_Enable	Enables the axis servo loop modes	733
AX_EnableProbe	Enables the storage of axes positions when the probing signal is generated	745
AX_Filter	Allows to configure the filters of an axis	743
AX_PLCToServo	Allows the PLC to reply the servo requests	763
AX_ReadParameter	Reads the parameters corresponding to an axis	708
AX_ReadPendingCommands	Allows to determine the status of axis inner commands	741
AX_ReadProbePosition	Reads probe values of the axes involve in the probing cycle	746
AX_ReadProbeStatus	Reads probe status of the axes specified	748
AX_ReadStatus	Reads axis status	736
AX_Reset	Is used to abort a command on the axis	735
AX_ResetPosition	Sets the axis position to zero	750
AX_SelectGearFoller	Allows to select the range for spindle axes or to set the axis servo error type	752
AX_ServoToPLC	Allows the servo to communicate with the PLC	761
AX_SetHardwareOvertravel	Sets the status of the hardware over-travel limit switches	730
AX_SetStatus	Allows to set some status flags of the axis	739
AX_SetTestMode	Sets the input or the out pin test mode for the axes specified	749
AX_SpindleSwitch	Allows to change the spindle use mode activating axis characterization	754
AX_WriteParameter	Changes the parameters corresponding to an axis	718
AXD_Command	Sends a digital drive command	700

AXD_ReadParameter	Reads a parameter of a digital drive	694
AXD_RTFeedbackRead	Reads the real-time variables of a digital drive	702
AXD_RTFeedbackStart	Executes the configuration for monitoring a digital drive variables	696
AXD_RTFeedbackStop	Stops reading digital drive variables	703
AXD_RTTTargetStart	Configures the writing process of the digital drive variables	704
AXD_RTTTargetStop	Stops the writing process of a digital drive	707
AXD_RTTTargetWrite	Writes the real time variables of a digital drive	706
AXD_WriteParameter	Writes a parameter of a digital drive	698
AXIS_ActivOffset	Requests the application of the offsets calculated for given axes	158
AXIS_CalcOffset	Requests to calculate the offsets to be applied for axes	160
AxisGet	Allows a process to get/to release axes resources.	120
AxisShareOFF	Release sharing axes of the process	33
AxisShareON	Request sharing axes of the process	32
CamAxiDefine	Defines an electronic cam associated with a master and slave axis.	200
CamAxiStart	Enables one or more electronic cams master/slave type	202
CamOutDefine	Defines an electronic cam associated with a master and to some inputs	205
CamOutStart	Enables the electronic cam output type	207
CamStop	Disables one or more active cams	209
CamUndefine	Deletes one or more input cams	210
CANopen_BoardCmd	Allows to stop and restart data transmission via the CANopen network	793
CANopen_GetEmcyInfo	Reads emergency conditions returned by a node that is part of the configuration	795
CANopen_GetEmcyNode	Makes it possible to test or return a node containing diagnostic data	798
CANopen_GetStatus	Returns some data regarding the status of the nodes included in the CANopen network	799
CANopen_NmtCmd	Sends NMT commands via the CANopen network	802
CANopen_ReadSdoCmd	Reads an object in the Object Dictionary of a node	804
CANopen_SyncCmd	Transmits a SYNC signal via the CANopen network	807
CANopen_WriteSdoCmd	Allows to write an object in the Object Dictionary of a node	808
ClearStrobeConsole	Resets (for the PLC only) the presence of a console event	125
ClearStrobeEmerg	Resets the presence (for PLC only) of an emergency	558
ClearStrobePProgram	Resets (for the PLC only) the presence of a partprogram event	138
CloseDB	Closes a database containing PLC data structures	845
ConsoleOFF	Disconnects an operator console from a process	601
ConsoleON	Connects an operator console from a process	602
ConsoleReadInput	Reads the status of console inputs	604
ConsoleReadOutput	Reads the status of console outputs	606
ConsoleSetup	Configures console selectors for a process in continuous mode	610
ConsoleSetupArray	Configures the operating modes of selectors	613
ConsoleWriteOutput	Writes the status of console outputs	608
CopyDB	Copies a database containing PLC data structures.	869

CpuUsage	Calculates the CPU usage percentage	846
CreateDB	Creates a database to store PLC data structures	871
CriticalStatus	Verifies if there are any WinPLUS tasks working in a critical zone	534
CycleStart	Request cycle start for a process	34
CycleStop	Request cycle stop for a process (release cycle start)	35
Decoder	Decodes some values in a word	518
Delay	Introduces a delay	531
DeleteDB	Deletes a database containing PLC data structures	847
DeleteRemapInput	Deletes forform memory all the inputs remap definitions	890
DeleteRemapOutput	Deletes forform memory all the remap definitions of the outputs	866
ECAT_CommStatus	Reads the EtherCAT communication status	652
ECAT_GetSlaveInfo	Reads of some information of an EtherCAT device in the fieldbus	660
ECAT_ReadDeviceData	Reads data about a device connected on the Sercos profile EtherCAT bus	659
ECAT_ReadDiagLog	Reads the notifications of the EtherCAT communication software	654
ECAT_SetSlaveInfo	Writes some informaton on an EtherCAT device on the fieldabus. The information can be configuration data or commands towards the device	664
ECAT_WriteDeviceData	Allows to write data about a device connected on the Sercos profile EtherCAT bus	656
ECAT_WriteMasterInfo	Sends some information to the master of the EtherCAT communication. The information can be configuration data or command towards the fieldbus	657
EmergencyStopClose	Closes the local E-STOP for the process communicated	542
EmergencyStopOpen	Opens the local E-STOP for the process communicated	543
Encoder	Encodes some bits in a word	519
EndOfBlockDisable	Disables the filter request on block execution end	106
EndOfBlockEnable	Enables the filter request on block execution end	107
EndReset	Notifies end of reset forform PLC	100
EnterCritical	Enters a critical zone with semaphore	532
EnterFeedHold	Requests to enter in the feedhold status	49
ExecBlock	Executes a part-program block	37
ExecMDIBlock	Executes a program block in mdi mode	36
ExitCritical	Leaves a critical zone with semaphore	533
ExitFeedHold	Requests the exit forform feedhold status	51
FastInputRead	Reads fast inputs	544
FastOutputRead	Reads fast outputs	545
FastOutputWrite	Allows to write fast outputs	546
FeedBypassOFF	Deactivate feed bypass	45
FeedBypassON	Activate feed bypass	44
FileListDB	Provides database list	873
FileReadWrite	Supplies functions living access to formatted files	547
Filter_Acc_Create	Creates a filter for the centripetal acceleration	618
Filter_Acc_Delete	Deletes an acceleration filter	629

Filter_Acc_Run	Allows to run an acceleration filter	619
Filter_Bessel_Create	Creates a Bessel acceleration filter	630
Filter_Bessel_Delete	Deletes a Bessel acceleration filter	621
Filter_Bessel_Run	Allows to run a Bessel acceleration filter	631
Filter_Boost_Create	Creates a start or reversal filter	622
Filter_Boost_Delete	Deletes a start or reversal filter	633
Filter_Boost_Run	Allows to run a start or reversal filter	623
Filter_LowPass1st_Create	Creates a first order low-pass.	634
Filter_LowPass1st_Delete	Deletes a first order low-pass.	625
Filter_LowPass1st_Run	Allows to run a first order low-pass.	635
Filter_Smooth_Create	Creates a floating average filter	626
Filter_Smooth_Delete	Deletes a floating average filter	637
Filter_Smooth_Run	Allows to run a floating average filter	627
ForceRemapInput	Allows to define a physical digital input remapping on a logic digital input.	848
ForceRemapOutput	Allows to define a physical digital output remapping on a logic digital output	875
FreeLibrary	This is used to download a DLL written by the user forform the address space of the calling process	839
GetAxisIDFromName	Determines the ID of an axis forform its name	584
GetAxisNameForFormID	Determines the name and the process of an axis	585
GetBoardInfo	Supplies a description of the axis board resources	573
GetDate	Reads the system date	841
GetJogIncrement	Reads jog increment value	140
GetReleaseInfo	Provides the software version installed in the CNC system	553
GetRemapAddress	Determines the remapping address of a physical or logical input/output	891
GetStrobeConsole	Allows to determine the console event present in the system	114
GetStrobeEmerg	Allows to determine the type of emergency present in the system	559
GetStrobePProgram	Allows to determine the partprogram an event present in the system	123
GetSystemInfo	Provides the CNC system type	577
GetSystemSerialNumber	Provides the Serial Number of the CNC system	554
GetTime	Reads the system time	842
GMC_ActivateOrigin	Activates an origin	410
GMC_ActivateResetQuote	Resets an axis	414
GMC_Bore	Executes boring cycle	440
GMC_CamAxiDefine	Defines an electronic cam associated to a master and to a slave axis	474
GMC_CamAxiStart	Enables one or more electronic cams mater/slave type	477
GMC_CamAxiStartOnPosition	Enables one or more electronic cams slave/master type, according to the value of the AxisId axis position	480
GMC_CamAxiStartOnSignal	Enables one or more electronic cams master/slave type on signal	484
GMC_CamOutDefine	Defines an electronic cam associated to a master to to the outputs.	488
GMC_CamOutStart	Enables an electronic cam output type	490

GMC_CamOutStartOnPosition	Enables an electronic cam output type	492
GMC_CamOutStartOnSignal	Enables one or more electronic cams master/slave type on signal	495
GMC_CamStop	Disables one or more active cams	497
GMC_CamStopOnPosition	Disables one or more active cams according to an axis position	501
GMC_CamStopOnSignal	Disables one or more cams active on signal	499
GMC_CamUndefine	Deletes one or more input cams	503
GMC_ChangePosLoop	Changes axes servo loop mode	384
GMC_ChangePosLoopALL	Changes all axes servo loop mode	386
GMC_Circle3D3P	Executes a circular motion for three given points	460
GMC_Circle3DCCW	Executes a circular counter-clockwise motion in space known the centre and the final point	454
GMC_Circle3DCW	Executes a circular clockwise motion in space known the centre and the final point	464
GMC_CircleCCW	Executes a counter-clockwise circular movement with a given centre	281
GMC_CircleCCWRadius	Executes a counter-clockwise circular movement with a given centre	287
GMC_CircleCW	Executes a clockwise circular movement with a given centre	293
GMC_CircleCWRadius	Executes a clockwise circular movement with a given centre	303
GMC_ContinuousAbort	Determines the ends of a continuous motion and its cancellation	309
GMC_ContinuousEnd	Determines the closure of a continuous stage and its cancellation	310
GMC_ContinuousParallel	Determines the closure of a continuous stage	311
GMC_ContinuousParam	Activates continuous parallel motions	315
GMC_ContinuousStart	Activated the continuous motion	316
GMC_CreateAxVirtual	Inserts the virtual axis among the axes managed by the interpolator	300
GMC_DeleteAxVirtual	Delete the virtual axis from the table of the axes managed by the interpolator	302
GMC_Drill	Executes drilling cycle	437
GMC_EnterFeedHold	Sets all PLC interpolators to feedhold	233
GMC_EnterHold	Sets all PLC interpolators to hold	235
GMC_ExitFeedHold	Ends the feedhold status for all PLC interpolators	234
GMC_ExitHold	Ends the hold status for all PLC interpolators	236
GMC_Homing	Executes the axis reset	423
GMC_HwndExe	Executes axes movement through hand wheel	319
GMC_HwndOFF	Disables axes movement through hand wheel	321
GMC_HwndON	Enables axes movement through hand wheel	322
GMC_HwndParam	Defines application parameters of the hand wheel on the axes	323
GMC_HwndParamExt	Defines the handwheel application parameters on the axes	324
GMC_InterpCreate	Creates an interpolator to be used by the PLC	226
GMC_InterpDelete	Removes a PLC interpolator	228
GMC_InterpEnterFeedHold	Sets an interpolator to feedhold status	229
GMC_InterpEnterHold	Requires an interpolator to enter the hold status	231
GMC_InterpExitFeedHold	Ends the feedhold status of an interpolator	234
GMC_InterpExitHold	Ends the hold status of an interpolator	232
GMC_InterpModify	Changes the configuration of the interpolator	237
GMC_InterpReadInfo	Reads the status of an interpolator	239

GMC_InterpReset	Asks a PLC interpolator to reset itself	242
GMC_Move	Executes a linear movement	244
GMC_MoveJog	Executes the manual movement Jog	248
GMC_MoveOnPosition	Executes a linear movement on the position of the axis	250
GMC_MoveOnSignal	Executes a linear movement conditional on the signal status	255
GMC_MoveRoundOFF	Executes a linear movement with release	269
GMC_MoveUntilSignal	Executes linear movement ending on signal	259
GMC_MoveUntilVariable	Executes a linear movement that ends on the basis of the value of a variable	273
GMC_MSLdefine	Executes the master/slave association	326
GMC_MSLexe	Immediately activated the master/slave following	328
GMC_MSLexeONposition	Activates mater/slave following on axis position	330
GMC_MSLexeONsignal	Activates the mater/slave following on signal	333
GMC_MSLfollow	Immediately modifies master/slave following ratio	335
GMC_MSLfollowONposition	Modifies the master/slave following ratio on axis position	337
GMC_MSLfollowONsignal	Modifies the master/slave following ration on signal	340
GMC_MSLjog	Immediately activates JOGGING for a slave	342
GMC_MSLjogONposition	Activates JOGGING for a slave on axis position	344
GMC_MSLjogONsignal	Activates JOGGING for a slave on axis	346
GMC_MSLparam	Defines the parameters of master/slave axes following	348
GMC_MSLrealign	Realigns the position of a Slave with reference to the Master	364
GMC_MSLreset	Immediately resets the value of the master compared to the slave	355
GMC_MSLresetONposition	Immediately resets the value of the master compared to the slave based on the axis position	357
GMC_MSLresetONsignal	Resets the master value compared to a slave based on a signal	360
GMC_MSLundef	Executes the dissociation between master/slave axes	362
GMC_Poly	Executes a polynomial interpolation	264
GMC_Probe	Executes probing cycle	365
GMC_ProtectAreaCmd	Works in a protected area previously defined	268
GMC_ProtectAreaDefine	Defines a protected area	370
GMC_ReadAxisInfo	Reads the status of an axis	372
GMC_ReadAxisInfoALL	Reads the status of all axes	378
GMC_Reset	Asks all PLC interpolators to reset themselves	243
GMC_SetAxisProperty	Activates the axes set up mode	388
GMC_SetAxisPropertyALL	Activates the axes set-up mode for all the axes	390
GMC_SetBitONposition	Sets a signal condition by axis position	392
GMC_SetBitONvariable	Sets a signal condition by value of a variable	394
GMC_SetDynamicOverride	Selects the Dynamic Override percentage and mode for specified axes	470
GMC_SetDynamicOverrideALL	Selects the Dynamic Override percentage and mode for axes associated to the PLC	472
GMC_SetFeed	Sets feederate on the profile	399
GMC_SetFeedOverride	Defines feedrate on the profile	396
GMC_SetFeedOverrideALL	Determines the feed % for the axes specified	398
GMC_SetJerk	Defines the S ramps JRK	400
GMC_SetOrigin	Configures an origin on axes	408
GMC_SetRamp	Changes the ramp mode for axes	401

GMC_SetRampALL	Modifies the ramp mode for all the axes	405
GMC_SetResetQuote	Immediately configures the axis reset value	412
GMC_SetResetQuoteONmarker	Configures axis reset on respect to the marker	416
GMC_SetResetQuoteONPosition	Configures the axis reset point as a function of axis position	417
GMC_SetResetQuoteONsignal	Configures the axis reset point influenced by a signal	419
GMC_SetResetQuoteShift	Configures the value of reset axis position change	421
GMC_SpindleChangeGear	Performs the gear change on the spindle	425
GMC_SpindleG96	Starts the spindle rotation at a certain constant cutting speed	426
GMC_SpindleG97	Starts the spindle rotation at a certain speed in RPM	428
GMC_SpindleOrient	Waits on axis position	430
GMC_Tapp	Executes tapping cycle without transductor	443
GMC_TappTransd	Executes tapping cycle with transductor	446
GMC_Thread	Executes threading cycle	450
GMC_ToleranceParams	Defines the tolerance for the calculation of circular trajectories	278
GMC_WaitONposition	Waits for value of a signal	432
GMC_WaitONsignal	Determines a certain waiting period	434
GMC_WaitONTIMEOUT	Waits for a value of a variable	435
GMC_WaitONvariable	Waits on axis position	436
HandWheelExe	Execute the axes movement through hand wheel	39
HandWheelOFF	Disable axes movement through hand wheel	41
HandWheelON	Enable the axes movement through hand wheel	38
HandWheelParam	Defines parameters for the application of the hand wheel on axes	42
HoldOFF	Requests the exit forform hold	47
HoldON	Requests to enter in hold	48
InitData	Initialises a number of adjacent variables in an indexed manner	520
InitMemory	Initialises a memory zone	524
IntToAscii	Converts a numerical value into an ASCII character	517
LoadLibrary	Function used to load a DLL written by the user into the address space of the calling process	837
LookUpT_Linear_Create	Calculates the coefficients of a linear interpolation	643
LookUpT_Linear_Delete	Deletes a look-up table with linear interpolation	638
LookUpT_Linear_Run	Allows to run an interpolation	645
LookUpT_Spline3_Create	Calculates a cubic spline coefficients	639
LookUpT_Spline3_Delete	Deletes a look-up table with linear interpolation	647
LookUpT_Spline3_Run	Allows to execute an interpolation	641
MasterSlaveDefine	Executes the association between a master axis and slave axes	52
MasterSlaveGO	Activates master slave following	59
MasterSlaveParam	Changes the master slave following ratio	60
MasterSlaveStatus	Can monitor the master slave following status	61
MasterSlaveSTOP	Deactivates the master slave following	62
MasterSlaveUndefine	Removes the master slave association	63
ME2_CommStatus	Reads the Mechatrolink I/II communication status	555
MoveData	Copies variables in indexed mode	526
MoveMemory	Copies data forform a memory area to another	528
OpenDB	Opens a database containing PLC data structures	852
OsWire_GetEmcyInfo	Reads data referred to emergencies on I/O nodes connected on OSWire Bus.	578

Pid_Create	Creates PID controller.	649
Pid_Delete	Deletes PID controller.	648
Pid_Run	Allows to run PID controller.	650
PP_BlockNumberSearch	Positions the pointer on the active part program of block number onwards or backwards	65
PP_BlockSearch	Moves the part program to the requested N block number	64
PP_ExecUntil	Selects end of the part-program execution	67
PP_ExecUntilBlockNumber	Selects a part program block as end point	66
PP_ExecUntilNext	Selects the part program execution stop and keeps the execution until the next condition	68
PP_GetBlockNumber	Determines the active part program block sequence numbers	70
PP_GetMarkers	Reads the active markers when a part program is running	71
PP.GetName	Determines the name of the active part program	73
PP_LineSearch	Places a pointer in the part program to the requested line	74
PP_LoadXmlMap	Loads a new XML remapping file	84
PP_MarkerSearch	Moves the part program to the required marker	72
PP_Resume	Resumes part program execution after mas	75
PP_RunInfoGO	Activates monitoring of a part program execution	76
PP_RunInfoRead	Reads monitoring data of a part program execution	77
PP_RunInfoStatus	Returns a part program execution monitoring status	79
PP_RunInfoSTOP	Disable monitoring of a part program execution	80
PP_Select	Activates a part program	81
PP_TextSearch	Finds a string inside the part program active	83
ProbeCreate	Creates a PLC diagnostic object	582
ProbeDelete	Deletes a PLC diagnostic object	580
ProbeOFF	Sets Probe signal at OFF.	583
ProbeON	Sets Probe signal at ON.	581
ProtectAreaDefine	Defines a protected area	85
ProtectAreaDisable	Removes a protected area	87
ProtectAreaEnable2D	Activates a protected area in 2d (only plane axes)	88
ProtectAreaEnable2D_2	Activates a protected area in 2d and ½ (with vertical axis)	90
PurgeRetainDB	Restores the retentive memory recovering spaces no more in use	879
ReadDB	Reads database record containing PLC data structures	584
ReadFilterConsole	Reads the data associated with a console event	110
ReadFilterEmerg	Reads the data associated to an emergency warning	561
ReadFilterPProgram	Reads the data associated with a pprogram event	129
ReadInputsIntoWord	Reads a series of inputs into a word	535
ReadProcessInfo	Reads the status of a process	92
ReadRemapInputCFG	Allows to upload in memory a file of digital inputs remapping definitions	880
ReadRemapOutputCFG	Allows to upload in memory a file of digital outputs remapping definitions	856
Refresh3DPositions	Refreshes the positions of 3D axes belonging	199
RemapInput	Executes digital inputs remapping	888
RemapInputByClass	Runs the inputs remapping according to their belonging class.	889
RemapOutput	Executes digital outputs remapping	864
RemapOutputByClass	Remaps the outputs according to their belonging class	865

RenameDB	Renames the database containing PLC data structures	893
Reset	Activates reset of a process	98
ResizeDB	Changes the number of records in a database containing PLC data structures	867
RS_AnalogRead	Reads a local A/D convertor on the axes board	781
RS_AnalogWrite	Writes a local D/A convertor on the axes board or on the OS-WIRE bridge device	782
RS_CreateResource	Allows to set a new D/A, A/D or transducer resource in the servo loop task	765
RS_DestroyResource	Deletes a D/A, A/D resources or a transducer previously set with the function RS_CreateResource	768
RS_SensorExecCmd	Executes commands on the sensor	770
RS_SensorReadPar	Reads sensor parameters	773
RS_SensorReadTable	Reads the sensor calibration table	777
RS_SensorWritePar	Writes sensor parameters	775
RS_SensorWriteTable	Writes the sensor calibration table	778
RS_TransducerRead	Reads the count of a local or remote encoder transducer on an OS-WIRE bridge	780
RunProgram	Allows to launch a program	840
S3_Command	Allows to run an IDN command of a device connected on SERCOSIII bus.	676
S3_CommStatus	Reads the SERCOSIII communication status.	683
S3_ReadDeviceData	Reads information about a device connected on SERCOSIII bus.	678
S3_ReadDialogLog	Reads the diagnostics internal to the communication master	684
S3_ReadParameter	Reads an IDN parameter of a device connected on the SERCOSIII bus.	679
S3_WriteDeviceData	Allows to write an IDN parameter of a device connected on the SERCOSIII bus.	628
S3_WriteParameter	Allows to write information about a device connected on SERCOSIII bus.	688
SearchData	Indexed search of a variable within a variables zone.	522
SelectAxis	Selects one or more axes for manual operations	101
SelectAxisAndOverride	Selects one or more axes to be moved manually and the associated override	102
SendMessage	Transmits a message on a specified queue	537
SendSemaphore	Activates a WinPLUS task waiting at a semaphore	539
SerialLineClose	Closes a work session for the serial line indicated in input	917
SerialLineOpen	Opens a work session for the serial line indicated in input	918
SerialLineReadSetup	Reads the configuration of a serial line	919
SerialLineReceive	Executes the reception of characters via the serial line specified	925
SerialLineReset	Function stops the activity of a serial line and resets the error condition	929
SerialLineStatus	Reads the status of a serial line	930
SerialLineTransmit	Transmits characters via the serial line specified	927
SerialLineWriteSetup	Is used to configure a serial line	922
SetAxisManualOverride	Selects override for the axes moved manually	104
SetDate	Allows to change the system date	843

SetDynamicAxOverride	Selects the percentage and the Override Mode for axes specified	178
SetDynamicOverride	Selects the percentage and the Dynamic Override mode for axes specified	198
SetFeedOverride	Select feed override percentage	147
SetFilterConsole	Specifies whether the PLC is or is not informed of console events generated in the system	108
SetFilterEmerg	Requests to inform the PLC about the system emergencies	566
SetFilterPProgram	Specifies whether the PLC is or is not informed of partprogram events generated in the system	127
SetJogIncrement	Sets jog increment	141
SetManualOverride	Selects manual feed rate override percentage	142
SetMultiBlockRetrace	Configures the retrace function of a program (mbr)	144
SetProcessMode	Sets mode of operation for the process	97
SetRapidOverride	Selects rapid feed override percentage	146
SetSearchInMemory	Defines the mode for the search in memory (rcm)	145
SetSpindleOverride	Selects spindle override percentage	148
SetTime	Writes the system time	844
SocketAccept	Accept an input connection request	898
SocketBind	Binds a local address to a socket	907
SocketClose	Closes the socket	900
SocketConnect	Tries to set a connection towards a certain destination address	909
SocketCreate	Creates a socket	901
SocketEnableNonblocking	Enables/disables a non-blocking mode on a socket.	911
SocketListen	Set a socket in connection requests listening status	903
SocketReceive	Receives data forform a socket	912
SocketSelect	Restores the socket status	905
SocketSend	Sends data	914
SOE_Command	Allows to execute an IDN command of a device connected on EtherCAT bus with SERCOS profile	669
SOE_ReadParameter	Reads an IDN parameter of a device connected on EtherCAT bus with SERCOS profile	666
SOE_WriteParameter	Allows to write an IDN parameter of a device connected on EtherCAT bus with SERCOS profile	671
SpindleGet	Requests the use of a spindle shared by PLC	149
SpindleRelease	Release the use of a spindle shared by the PLC	150
TBL_Lock	Requests exclusive access to a table	586
TBL_ReadField	Reads a single field in a table record	588
TBL_ReadRecord	Reads a complete table record	594
TBL_SearchField	Allows to search a single field in a table	598
TBL_Unlock	Gives exclusive access to a table	587
TBL_WriteField	Allows to write a single field in a table record	591
TBL_WriteRecord	Writes a complete table record	596
TeachPendOFF	Disables the use of the serial teach pendant for a certain process	151
TeachPendON	Enables the use of the serial teach pendant for a certain process	152
TOOL_ActivOffset	Requests activation of a tool offset	153

TOOL_CalcOffset	Calculates, on the basis of offset, the offsets to be applied on the axes	155
TOOL_GetSlaveID	Sets the slave tools	132
ToolCenterPointCFGRead	Reads the TCP configuration for a given process	190
ToolCenterPointCFGWrite	Updates the TCP configuration for a given process	180
ToolCenterPointOFF	Disables TCP for a given process	195
ToolCenterPointON	Enables the TCP use for a process	194
ToolCenterPointRead	Restores the TCP operative status for a given process	188
ToolCenterPointRestore	Restores the last TCP enabled for a given process	186
ToolCenterPointStatus	Restores the TCP operative status for a given process	196
ToolCenterPointWrite	Updates the operative TCP status for a given	184
TranslateError	Returns the message corresponding to an error code	570
TranslateErrorExt	Returns the message corresponding to a specific error code	571
VAR_Read	Reads a process variable	163
VAR_ReadArray	Reads a process variable	165
VAR_ReadEx	Reads a process variable	173
VAR_Write	Writes a process variable	167
VAR_WriteArray	Writes a process variable	171
VAR_WriteEx	Writes a process variable	175
VAR_WriteProtect	Writes a Write Protect type process variable	169
WaitOnMessage	Wait to receive a message	565
WaitOnSemaphore	Puts a WinPLUS task in wait at semaphore status	540
WaitOnSemaphoreStatus	Determines whether there are any WinPLUS tasks waiting at a semaphore	541
WarningMessage	Displays a warning message	538
WarningMessageDisplay	Displays a series of warning messages by turns	568
WinPlusEmergency	Generates a non-recoverable WinPLUS emergency	560
WriteDB	Write a database record containing PLC data structures	895
WriteOutputsForFormWord	Writes a set of outputs forform a Word	536
XMLClose	Closes an XML file	812
XMLOpen	Opens an XML file	813
XMLRead	Reads one or more data forform the elements of an XML file	815
XMLReadAttribute	Reads one or more data forform the attributes of an XML file	819
XMLReadAttributeEx	Reads one or more data of the XML file attributes	822
XMLReadEx	Reads one or more data of the XML file elements	825
XMLRemove	Removes an element forform an XML file	831
XMLRemoveAttribute	Removes an attribute forform an XML file	833
XMLSave	Saves an XML file in files system	835
XMLWrite	Writes one or more data in the elements of an XML file	817
XMLWriteAttribute	Writes one or more data in the attributes of an XML file	828

2. OPEN_CNC LIBRARY: Functions for communication with CNC processes



The functions of the OPEN_CNC library are related to the OPEN20 and OPEN30 models.

AxisShareON	Requests sharing axes of the process
AxisShareOFF	Release sharing axes of the process
CycleStart	Requests cycle start for a process
CycleStop	Requests cycle stop for a process (release cycle start)
ExecMDIBlock	Executes a program block in mdi mode
ExecBlock	Executes a part-program block
HandWheelON	Enables the axes movement through hand wheel
HandWheelExe	Executes the axes movement through hand wheel
HandWheelOFF	Disables axes movement through hand wheel
HandWheelParam	Defines parameters for the application of the hand wheel on axes
FeedBypassON	Activate feed bypass
FeedBypassOFF	Deactivate feed bypass
ActiveReset	Requests the active reset function from the process
HoldOFF	Requests the exit from hold
HoldON	Requests to enter in hold
EnterFeedHold	Requests to enter in the feedhold status
ExitFeedHold	Requests the exit from feedhold status
MasterSlaveDefine	Executes the association between a master axis and slave axes
MasterSlaveGO	Activates master slave following
MasterSlaveParam	Changes the master slave following ratio
MasterSlaveStatus	Can monitor the master slave following status
MasterSlaveSTOP	Deactivates the master slave following
MasterSlaveUndefine	Removes the master slave association
PP_BlockSearch	Moves the part program to the requested N block number
PP_BlockNumberSearch	Positions the pointer on the active part program of block number onwards or backwards
PP_ExecUntilBlockNumber	Selects a part program block as end point
PP_ExecUntil	Selects where the part-program execution ends.
PP_ExecUntilNext	Selects the part program execution stop and keeps the execution until the next condition.
PP_GetBlockNumber	Determines the active part program block sequence numbers
PP_GetMarkers	Reads the active markers when a part program is running
PP_MarkerSearch	Moves the part program to the requested marker.
PP_GetName	Determines the name of the active part program
PP_LineSearch	Moves the part program to the requested line.
PP_Resume	Resumes part program execution after mas
PP_RunInfoGO	Activates monitoring of a part program execution
PP_RunInfoRead	Reads monitoring data of a part program execution

PP_RunInfoStatus	Returns a part program execution monitoring status
PP_RunInfoSTOP	Disable monitoring of a part program execution
PP_Select	Activates a part program
PP_TextSearch	Finds a string inside the part program active
ProtectAreaDefine	Define a protected area
ProtectAreaDisable	Remove a protected area
ProtectAreaEnable2D	Activate a protected area in 2d (only plane axes)
ProtectAreaEnable2D_2	Activate a protected area in 2d and ½ (with vertical axis)
ReadProcessInfo	Reads the status of a process
SetProcessMode	Sets mode of operation for the process
Reset	Activates reset of a process
EndReset	Notifies end of reset from PLC
SelectAxis	Selects one or more axes for manual operations
SelectAxisAndOverride	Selects one or more axes to be moved manually and the associated override
SetAxisManualOverride	Selects override for the axes moved manually
EndOfBlockdDisable	Disables the filter request on block execution end
EndOfBlockEnable	Enables the filter request on block execution end
SetFilterConsole	Specifies whether or not the PLC is informed of console events generated in the system
ReadFilterConsole	Reads the data associated with a console event
GetStrobeConsole	Determines the console event present in the system
AckStrobeConsole	Informs the system that a console event has been acquired by the PLC
ClearStrobeConsole	Resets (for the PLC only) the presence of a console event
SetFilterPProgram	Specifies whether the PLC is or is not informed of partprogram events generated in the system
ReadFilterPProgram	Reads the data associated with a pprogram event
GetStrobePProgram	Determines the partprogram event present in the system
AckStrobePProgram	Informs the system that a partprogram event has been acquired by the PLC
ClearStrobePProgram	Resets (for the PLC only) the presence of a partprogram event
GetJogIncrement	Reads jog increment value
SetJogIncrement	Sets jog increment
SetManualOverride	Selects manual feed rate override percentage
SetMultiBlockRetrace	Configures the retrace function of a program (mbr)
SetSearchInMemory	Defines the mode for the search in memory (rcm)
SetRapidOverride	Selects rapid feed override percentage
SetFeedOverride	Select feed override percentage
SetSpindleOverride	Selects spindle override percentage
SpindleGet	Requests the use of a spindle shared by PLC
SpindleRelease	Release the use of a spindle shared by the PLC
TeachPendOFF	Disables the use of the serial teach pendant for a certain process
TeachPendON	Enables the use of the serial teach pendant for a certain process
TOOL_ActivOffset	Requests activation of a tool offset
TOOL_CalcOffset	Calculates, on the basis of offset, the offsets to be applied on the axes
AXIS_ActivOffset	Requests the application of the offsets calculated for given axes
AXIS_CalcOffset	Requests to calculate the offsets to be applied for axes
TOOL_GetSlaveID	Sets the slave tools

VAR_Read	Reads a process variable
VAR_ReadArray	Reads a process variable
VAR_Write	Writes a process variable
VAR_WriteProtect	Writes a process variable Write Protect type
VAR_WriteArray	Writes a process variable
VAR_ReadEx	Reads a process
VAR_WriteEx	Writes a process
AbortTappingCycle	Aborts the tapping cycle
SetDynamicAxOverride	Selects the Dynamic Override percentage and mode for the specified axes.
ToolCenterPointCFGWrite	Updates the TCP configuration for a given process.
ToolCenterPointWrite	Updates the TCP operating status for a given process
ToolCenterPointRestore	Restores the last TCP enabled for a process
ToolCenterPointRead	Restores the TCP operating status for a process
ToolCenterPointCFGRead	Reads a TCP process configuration
ToolCenterPointON	Enables TCP for a given process
ToolCenterPointOFF	Disables TCP for a given process
ToolCenterPointStatus	Restores the TCP operating status for a process
SetDynamicOverride	Sets Dynamic Override percentage and mode
Refresh3DPositions	Refreshes the 3D axes positions in a process
CamAxiDefine	Defines an electronic cam associated with a master and slave axis..
CamAxiStart	Enables one or more electronic cams master/slave type
CamOutDefine	Defines an electronic cam associated with a master and to some inputs
CamOutStart	Enables the electronic cam output type
CamStop	Disables one or more active cams
CamUndefine	Deletes one or more input cams

2.1 AxisShareON

Function AxisShareON requests to share the axes.

Syntax:

```
ret_code := AxisShareON (ExecMode, ExecStatus, Process, AxisId);
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Execution status
<i>Process</i> (INT)	Process number [1..24]
<i>AxisId</i> (INT)	Axes identifier [1..64]

Incidental overloads:

AxisShareON (INT,DWORD,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This function allows the axes sharing between a process and the PLC. It can be activated only if the process is in IDLE, MBR and MAS modes.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example:

```
Ret : dword;
(*It shares axes with ID 1 and 2 belonging to process 1*) Ret := AxisShareON
(MODE_NOWAIT, ULO, 1, 1, 2);
```

See also:

AxisShareOFF

2.2 AxisShareOFF

Function AxisShareOFF releases the axes sharing.

Syntax:

ret_code := AxisShareOFF (*ExecMode*, *ExecStatus*, *Process*, *AxisId*) ;

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Execution status
<i>Process</i> (INT)	Process number [1..24]
<i>AxisId</i> (INT)	Axes identifier [1..64]

Incidental overloads:

AxisShareOFF (INT,DWORD,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This function releases the axes previously shared between a process and the PLC. It can be activated only if the process is in IDLE, MBR and MAS modes.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example:

```
Ret : dword;
(* It releases axes with ID 1 and 2 belonging to process 1*)
Ret := AxisShareOFF (MODE_NOWAIT, UL0, 1, 1, 2);
```

See also:

AxisShareON

2.3 CycleStart

Function CycleStart requests a Cycle Start for the process.

Syntax:

```
ret_code := CycleStart (ExecMode, ExecStatus, Process) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function is used to simulate the front panel's cycle start button in WinPLUS. In the case of execution in MODE_NOWAIT mode, the function immediately returns to the PLC and there is no synchronisation with the execution of the CYCLE; process status has to be monitored in order to determine the outcome of command execution. In MODE_WAIT mode, the function returns to the PLC at the end of the CYCLE command, which depends on process status and the execution mode set on the process. In WAIT_ACCEPT mode, the function returns to the PLC the moment the CYCLE command is put into execution and hence it has been accepted by the process and activated. In MODE_NOWAIT_ACK mode the function returns to the PLC immediately; with variable *ExecStatus* it is possible to verify the execution status of the command: if it is !6#80000000 the command is being executed, if it is 0, the command has been completed successfully, other values mean that the command has ended with an error.

Return values

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example:

```
Ret      : dword;
CycStatus : dword;
Ret := CycleStart (MODE_NOWAIT_ACK, CycStatus , 1) ;
```

See also:

CycleStop, Reset, HoldON, HoldOFF, ReadProcessInfo

2.4 CycleStop

Function CycleStop requests cycle stop for a process (cycle start released).

Syntax:

```
ret_code := CycleStop (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function stops the axis motion related to the process specified.

Return values:

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example:

```
Ret      : dword;
StoStatus : dword;
Ret := CycleStop (MODE_NOWAIT_ACK, StoStatus , 1) ;
```

See also:

CycleStart, Reset, HolON, HoldOFF, ReadProcessInfo

2.5 ExecMDIBlock

Function ExecMDIBlock executes a program block in MDI mode.

Syntax:

```
ret_code := ExecMDIBlock (ExecMode, ExecStatus, Process, Command) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Command (STRING) Command/Block to be executed

Execution mode:

Wait / NoWait

Use:

This function executes a Part Program block in MDI mode. The block to be executed must be entered in string form in variable Command. This function causes the immediate execution of the block without having to force a cycle start, since this is already executed by the function itself.

Function can be activated only if:

- › The process is in MDI;
- › The process is in IDLE or HOLD or MAS status
- › The string that is copied onto the MDI buffer must be a valid part program block. If there is a Paramacro execution request in the block, it is necessary to switch to AUTO or SEMIAUTO and then give a CycleStart command to execute the complete command. This functionality is not valid in HOLD and MAS status;

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example:

```
Ret      : dword;
MdiStatus : dword;
Ret := ExecMDIBlock (MODE_NOWAIT_ACK, MdiStatus , 1, 'GXYZ') ;
```

See also:

[ReadProcessInfo](#)

2.6 ExecBlock

The function ExecBlock executes a part-program block.

Syntax

```
ret_code := ExecBlock (ExecMode, ExecStatus, Process, Command) ;
```

Input parameters

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>Process (INT)</i>	Process number [1..24]
<i>Command (STRING)</i>	Command/Block to execute

Execution Mode

Wait / NoWait

Use

This function allows the execution of a Pert program block.

The block to be executed should be transformed in string format in the Command variable. This function enables the immediate block execution with no request for CycleStart as this is already executed by the function itself.

In the command, a Paramacro or a subroutine execution can be required (for example asking for the execution of a block containing a CLT)

The function is enabled only if:

- ▷ the process is in HOLD or MAS status;
- ▷ the string (to be copied in the MDI buffer) must be a correct block of part-program

Return Values

The following table lists all the values the *ret_code* variable can have

<i>ret_code (HEX)</i>	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret      : dword;
Status   : dword;

Ret := ExecBlock (MODE_NOWAIT_ACK, Status , 1, 'GXYZ') ;
```

See also

[ReadProcessInfo](#)

2.7 HandWheelON

Function HandWheelON enables the axes move by hand wheel.

Syntax:

```
ret_code := HandWheelON (ExecMode, ExecStatus, Process, AxisId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Execution mode command [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Execution status command
<i>Process</i> (INT)	Process number [1..24]
<i>AxisId</i> (INT)	Axes identifier to be enabled for hand wheel [1..64]

Possible overloads:

HandWheelON (INT,DWORD,INT,INT[,INT],...)

Execution mode:

Wait / NoWait

Use:

This instruction enables the hand wheel motion for up to 12 axes. Axis/hand wheel configuration and paring, using the function *HandWheelParam*, make the instruction operative. The instruction can be send at any time.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Result : dword;
(* Axes 1 and 2 enabled to be moved by hand wheel *)
Result := HandWheelON (MODE_WAIT, UL2, 1, 1, 2 );
```

See also:

[HandWheelExe](#), [HandWheelParam](#), [HandWheelOFF](#)

2.8 HandWheelExe

Function HandWheelExe moves axes by hand wheel.

Syntax:

```
ret_code := HandWheelExe (ExecMode, ExecStatus, Process, Axis) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>Process (INT)</i>	Process number [1..24]
<i>Axis (AXISHwndExe_Struct)</i>	Axes to be moved by hand wheel

Possible overload:

HandWheelExe (INT,DWORD,INT, AXISHwndExe_Struct[,AXISHwndExe_Struct]...)

Execution mode:

Wait / NoWait

Use:

This instruction executes the hand wheel motion for the axes of the process. Up to 12 structures (AXIS_HwndExe_Struct) can be associated with the function in order to perform several hand wheel moves at the same time.

The structure of the axes move parameters is:

AxisId	INT	Process axis identifier to be associated to the hand wheel.
Tick	INT	Hand wheel motion data refresh time.
Pitch	INT	Pulses per hand wheel revolution.

The move signals the hand wheel *Steps* number to take signalling enables the motion, converting this value in distance according to the parameters used during the axis/hand wheel (*HandWheelParam*) association, that is:

$$\text{Space} = \text{Steps} * \text{Scale} / \text{Pitch}$$

The value obtained will be added to the spaces previously signalled, in fact, PLC will cyclically notify the number of Steps to cover. *Tick* identifies the cycle time used by the PLC to refresh the steps. This parameter calculates the right axis speed rate limited by axis manual moves. A reverse motion stops the axis at null feed restarting its move in the opposite direction; the space not covered in the original direction is lost. If axis is homed, software over travel limits will be activated during the hand wheel motion.

The instruction can be sent at any time. To move the axis in hand wheel mode, HANDWHEEL (see *SetProcessMode*) must be active and motions have to be started (see *CycleStart*) A specific instruction has to be sent to stop the motion (see *CycleStop*).

The correct sequence instructions to move the axes are:

<i>HandWheelParam</i>	Hand wheel configuration and its axis association.
<i>HandWheelON</i>	Axes to be moved by hand wheel is enabled.
<i>SetProcessMode</i>	Hand wheel enabled.
<i>SelectAxis</i>	Selection of the axes to be moved.
<i>CycleStart</i>	Move started.
<i>HandWheelEXE</i>	Move executed.
<i>CycleStop</i>	Move ended.
<i>HandWheelOFF</i>	Disable hand wheel axis movement.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Result : dword;
AxisWhlE1    : AXISHwndExe_Struct;
AxisWhlE2    : AXISHwndExe_Struct;

AxisWhlE1.AxisId := 1;
AxisWhlE1.Tick   := 50;
AxisWhlE1.Pitch  := 32;
AxisWhlE2.AxisId := 2;
AxisWhlE2.Tick   := 50;
AxisWhlE2.Pitch  := 32;
Result := HandWheelExe (MODE_WAIT, UL2, 1, AxisWhlE1, AxisWhlE2 );
```

See also:

[HandWheelOFF](#), [HandWheelParam](#), [HandWheelON](#), [SetProcessMode](#), [SelectAxis](#), [CycleStart](#), [CycleStop](#)

2.9 HandWheelOFF

Function HandWheelOFF disables the hand wheel move.

Syntax:

ret_code := HandWheelOFF (*ExecMode*, *ExecStatus*, *Process*, *AxisId*) ;

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>AxisId</i> (INT)	Axes identifier to disable hand wheel mode 1..64]

Possible overload:

HandWheelOFF (INT,DWORD,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction disables hand wheel movement for up to 12 axes of the function. The instruction can be sent at any time; if hand wheel mode is active, the axes will stop at zero speed and all the space left (*HandWheelExe*) will be lost.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Result : dword;
(* Hand wheel disabled for axes 1 and 2 *)
Result := HandWheelOFF (MODE_WAIT, UL2, 1, 1, 2 );
```

See also:

[HandWheelExe](#), [HandWheelParam](#), [HandWheelON](#)

2.10 HandWheelParam

The HandWheelParam function refers to the hand wheel application parameters on the axes.

Syntax:

```
ret_code := HandWheelParam (ExecMode, ExecStatus, Process, Axis) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Axis</i> (AXISHwndParam_Struct)	Axes to configure in hand wheel mode

Possible overload:

HandWheelParam (INT,DWORD,INT, AXISHwndParam_Struct[,AXISHwndParam_Struct]...)

Execution mode:

Wait / NoWait

Use:

This instruction associates the values enabling the hand wheel move to the axes. Up to 12 axes (AXISHwndExe_Struct) can be configured and the instruction can be sent at any time. After this instruction, the function HandWheelON must be activated to enable the axis hand wheel move.

The structure to configure the axes is:

Field	Type	Description
AxisId	INT	Process axis ID to associate with hand wheel.
Scale	LREAL	Distance to cover with one hand wheel turn. Value is in mm/revolution or inches/revolution.
Pitch	INT	Pulses for one hand wheel turn.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Result : dword;  
  
AxisWhlP1    : AXISHwndParam_Struct;  
AxisWhlP2    : AXISHwndParam_Struct;  
  
AxisWhlP1.AxisId := 1;  
AxisWhlP1.Scale  := 5.0;  
AxisWhlP1.Pitch   := 32;  
AxisWhlP2.AxisId := 2;  
AxisWhlP2.Scale  := 10.0;  
AxisWhlP2.Pitch   := 32;  
  
Result := HandWheelParam (MODE_WAIT, UL2, 1, AxisWhlP1, AxisWhlP2 );
```

See also:

[HandWheelOFF](#), [HandWheelExe](#), [HandWheelON](#)

2.11 FeedBypassON

Function FeedBypassON activates feed bypass.

Syntax:

```
ret_code := FeedBypassON (ExecMode, ExecStatus, Process, Velocity) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Velocity (LREAL) Velocity

Execution mode:

Wait / NoWait

Use:

Feed rate bypass is a feature which can instantaneously change the feed rate to any desired value within the limits of the axes. This function can be used for fast approach to the work piece ("gap elimination"). Express the desired feed rate in configured units, i.e. if the machine was configured in inches, the **velocity** parameter has to be a feed rate expressed in inches per minute. If the feed rate **velocity** is higher than the configured rapid feed rate of the axes involved, the system will limit the resulting feed rate to the rapid of the slowest axis involved.

This function can be activated during the linear point-to-point interpolation (G01 with G29) with G94 or G95 feed rate coding. In all other interpolation modes the active feed rate bypass mode will be aborted or the request with FeedByPassOn will be ignored. An ignored request will not be memorised. Changing the process status will also lead to the cancellation of feed rate bypass; i.e. when the process goes IDLE, the feed bypass will be switched off automatically. During feed rate bypass, the feed rate override selector can still be used to fine tune the bypass speed. Use the FeedByPassOff system function call to restore the original feed rate as programmed in the part program.



The effect of the feed rate bypass call is instantaneous. There is no synchronization with the part program blocks or the motion.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;
Dummy    : dword;
Ret := FeedBypassON (MODE_NOWAIT, Dummy, 1, 1000.0) ;
```

See also:

[FeedBypassOFF](#), [ReadProcessInfo](#)

2.12 FeedBypassOFF

Function FeedBypassOFF disables feed bypass.

Syntax:

```
ret_code := FeedBypassOFF (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function will instantaneously restore the programmed feed rate when feed rate bypass is in effect. Use this system function call to complete the quick approach to the work piece.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;
Dummy    : dword;
Ret := FeedBypassOFF (MODE_NOWAIT, Dummy, 1) ;
```

See also:

[FeedBypassON](#), [ReadProcessInfo](#)

2.13 ActiveReset

The ActiveReset function requests the Active Reset function from the process.

Syntax:

```
ret_code := ActiveReset (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode

Wait / NoWait

Use

Active reset is a non-modal function. It can only be called in AUTO or BLOCK/BLOCK mode under following conditions:

- ▷ a part program must be active
- ▷ the system must be in hold or the program never started or the system must be in an end of block stop.

Active reset will cause following actions:

- ▷ the next blocks in sequence will be set up again, assuming the actual position as the start position
- ▷ the next block becomes the active block
- ▷ the hold will be released (if it was on) S_HOLDA will be false S_RUNH will be true

If the function active reset is invoked during MDI mode, the active block will be cleared from the CRT and the internal buffer, if in hold, hold will be released.

Active reset does not have influence on any offsets, G codes and auxiliary functions.

There are many cases in which an active reset may lead to an error. Please consult the programming manual for proper use of active reset.

See also:

WinPLUS Application Manual for a detailed explanation of active reset

Return values

The following table gives the possible values of variable *ret_code*:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example

```
Ret    : DWORD ;
Dummy  : DWORD ;
Ret := ActiveReset (MODE_WAIT, Dummy, 1) ;
```

2.14 HoldOFF

Function HoldOFF requests the exit from hold.

Syntax:

```
ret_code := HoldOFF (ExecMode, ExecMode, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function makes the interpolator exit from HOLD status function call.

Return values:

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example

```
Ret      : dword;
HldStatus : dword;
Ret := HoldOFF (MODE_NOWAIT_ACK, HldStatus , 1) ;
```

See also:

[CycleStop](#), [Reset](#), [CycleStart](#), [HoldON](#), [ReadProcessInfo](#)

2.15 HoldON

The function HoldON requests to enter in hold mode.

Syntax:

```
ret_code := HoldON (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode

Wait / NoWait

Use:

This function stops the interpolator of the requested process. In this status of the NC, it is possible to execute manual motion, to execute MDI motion and to invoke multi-block-retrace.

Return values:

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;
```

```
HldStatus : dword;
```

```
Ret := HoldON (MODE_NOWAIT_ACK, HldStatus , 1) ;
```

See also:

[CycleStop](#), [Reset](#), [CycleStart](#), [HoldOFF](#), [ReadProcessInfo](#)

2.16 EnterFeedHold

Function EnterFeedHold requests to enter the FeedHold status.

Syntax:

```
ret_code := EnterFeedHold (ExecMode, ExecMode, Process, OnEndOfBlock) ;
```

Input parameters

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

OnEndOfBlock (BOOL) Request to enter on end of block

Execution mode:

Wait / NoWait

Use:

This function simply sets the feed rate to a zero value. Any on-going interpolating axes motion in the selected process is suspended. The motion can be resumed with ExitFeedHold.

If *OnEndOfBlock* is true, a stop at velocity 0 occurs as the axis reaches the end of the movement block underway.

- ▷ This function is not executed when the control is working in G33 (thread cutting) or G84 (tapping).
The *ret_code* variable contains a corresponding error code.
- ▷ The feedhold state is not cancelled by a RESET.

In the event of stop at end of movement:

- ▷ This function makes sense only during continuous movements; in all other types of motion, in fact, the end point is reached at zero velocity and hence this function is an intrinsic characteristic of the motion itself. At any rate, at the end of all types of motion, the feed rate is set to zero.
- ▷ This function does not ensure that the movement will stop at its end point, if the stop command is executed at a certain distance from the end point of the movement, and this distance is not sufficient to stop the movement (account duly taken of the accelerations configured on the axes), in fact, then the stop occurs during the subsequent movement(s); the behaviour will be the same as in a normal FeedHold.
- ▷ If a stop at end point has been correctly requested, the stopping position may differ from the end point of a movement by a distance of ca 0.01 mm.

Return values

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example

```
Ret      : dword;
```

```
Dummy    : dword;
```

```
Ret := EnterFeedHold (MODE_NOWAIT, Dummy, 1, false) ;
```

See also

[ExitFeedHold](#), [ReadProcessInfo](#)

2.17 ExitFeedHold

The function ExitFeedHold requests to come out of the FeedHold status.

Syntax:

```
ret_code := ExitFeedHold (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function makes the interpolator exit the feedhold state. The speed rate will be restored at the same value previous to the entry in feedhold state. If this function is called when the interpolator is not in feedhold state, nothing happens.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret      : dword;
Dummy    : dword;
Ret := ExitFeedHold (MODE_NOWAIT, Dummy, 1) ;
```

See also

[EnterFeedHold](#), [ReadProcessInfo](#)

2.18 MasterSlaveDefine

This function MasterSlaveDefine executes the association between a master axis and its slaves.

Syntax:

```
ret_code := MasterSlaveDefine (ExecMode, ExecStatus, Process, Mode, FollRate, Space,  
                  MasterId, SlaveId) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Mode</i> (INT)	Following mode
<i>FollRate</i> (LREAL)	Following ratio
<i>Space</i> (LREAL)	Synchronisation distance
<i>MasterName</i> (INT)	Id axis Master
<i>SlaveName</i> (INT)	Id axis Slave

Possible overload:

MasterSlaveDefine(INT,DWORD,INT,INT,LREAL,LREAL,INT,INT,...)

Execution mode:

Wait / NoWait

Use:

This function requests the association between a master axis and slave axes. This instruction **DOES NOT activate the following function**, which is activated by a specific command. This means that after this instruction a movement of the master does not bring about a movement of the slave(s). The command is accepted only if the process is in IDLE.

The master axis can identify either an axis present in the process in which the command is activated or an axis which is not present; in the latter case, a “virtual” axis, having the name specified, will be created; this axis will have the dynamic characteristics inherited from the slave axes (the lowest values of feed rate, accelerations and jerk). This axis may be part of a virtualisation (UPR, UDA,...) and also of a TCP. The master axis Homing operation cannot be performed.

The slave axis may be an axis present in the process in which the command is activated or another axis belonging to the PLC or to another process (in this case the axis must be SHARED); the slave may however be a SHARED axis, i.e. an axis shared with the machine logic environment. In this connection, the axis may continue to be moved by the PLC even after the association with the master, however it cannot be moved while it is following the master. It cannot be part of any virtualisation or TCP. It can be defined up to 17 slave axes among which 6 at maximum cannot belong to that specific process.

The *Mode* parameter defines the master axis following mode by a slave axis. It may be:

Mode	Mnemonic	Description
0	MODE_COPY	The slave follows the master point by point
1	MODE_VELO	The slave follows the master in velocity mode
2	MODE_SPC	The slave follows the master in position mode
3	MODE_SPCVEL	The slave follows the master in position mode with synchronisation distance recovery
4	MODE_SPCVEL2	The slave follows the master in position mode with synchronisation and released distance recovery

Parameter *FollRate* defines the master axis following rate on the part of the slave axis; this value must be regarded as a multiplication factor for the velocity or the distance covered by the master. With a value of 1.0 the slave copies the motion of the master exactly; with values smaller than 1.0 it reduces it, with values greater than 1.0 it increases it. This value may be preceded by a sign.

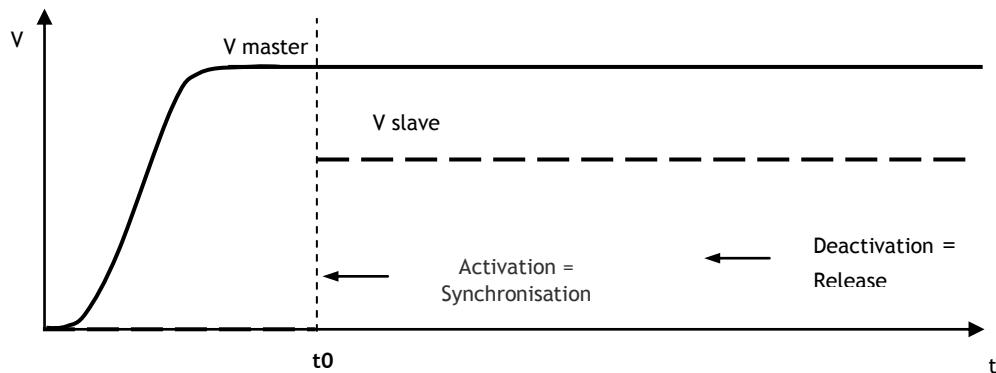
The *Space* parameter defines the distance to be covered by the slave to synchronise with the motion of the master.

Mode_Copy

In this mode, the slave axis follows the master proportionately to the value of the ratio (if the ratio = 1, the slave reproduces the movement of the master axis exactly), synchronisation is instantaneous and the variation in the feed rate of the slave is “in steps”. Slave position and feed rate values are calculated, instant by instant, according to the following formulas:

$$V_{slave} = V_{master} * FollowRate$$

$$PosSlave = PosSlave_{t0} + (PosMaster - PosMaster_{t0}) * FollowRate$$



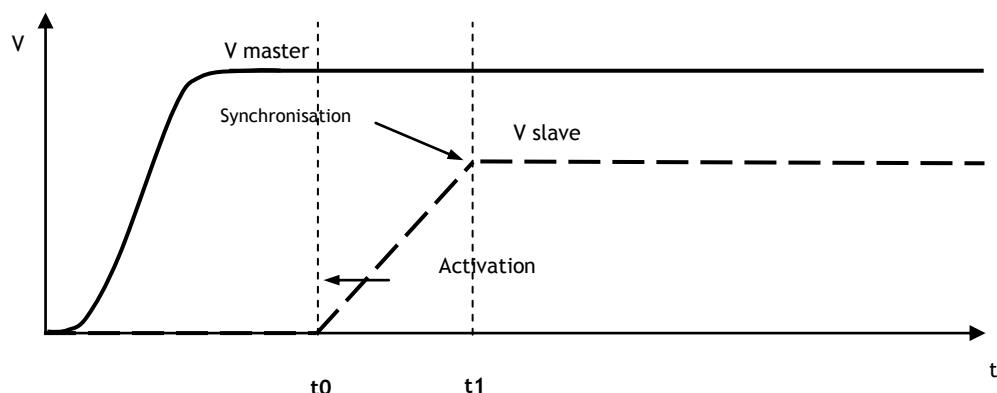
If the speed specified for the slave axis exceeds the maximum admissible value for this axis, the system will reduce the feed rate requested accordingly and will give out an emergency (servo error) message, in that the slave is unable to follow the required position.

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed.

Mode_Velo

In this mode, the slave follows the feed rate of the master proportionately to the value of the ratio (if the ratio = 1, the slave copies the movement of the master axis exactly); the synchronisation depends on the dynamic characteristics of the slave axis and/or the Space parameter which defines the synchronisation distance.

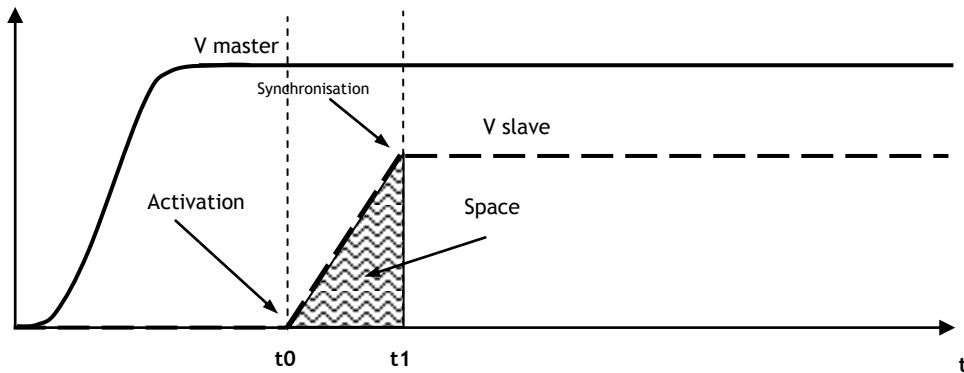
If the distance = 0, the slave will synchronise with the master based on its maximum acceleration and using linear ramps **only**.



If the value is not 0, the slave will synchronise with the master based on an acceleration calculated as a function of the synchronisation distance and using linear ramps **only**. The acceleration will be calculated again with each sampling process based on the following formula :

$$A_{slave} = ((V_{master} * FollowRate)^2 + V_{slave}^2) / 2 * Space$$

where the value of the distance is gradually reduced based on the distance covered during the synchronisation stage. No check is made on the ensuing acceleration value, and therefore servo errors may arise if the acceleration exceeds the maximum value that can be withstood by the axis.



Once the synchronisation with the master has taken place, the slave will move according to this formula:

$$V_{slave} = V_{master} * FollowRate$$

The feed rate (V_{slave}) determined in this manner is “theoretical”, since it is necessary to determine whether this request is compatible with the dynamic characteristics of the axis (maximum feed rate and maximum acceleration). The moment the feed rate of the master varies, the slave will follow this variation based on its own acceleration value. If the feed rate requested of the slave exceeds its maximum admissible feed rate, the system will reduce the feed rate requested accordingly. Hence, the feed rate and acceleration values with which the slave has to be moved, V_{slave} , and A_{slave} , will be determined instant by instant.

The position of the slave will therefore be calculated on the basis of these values:

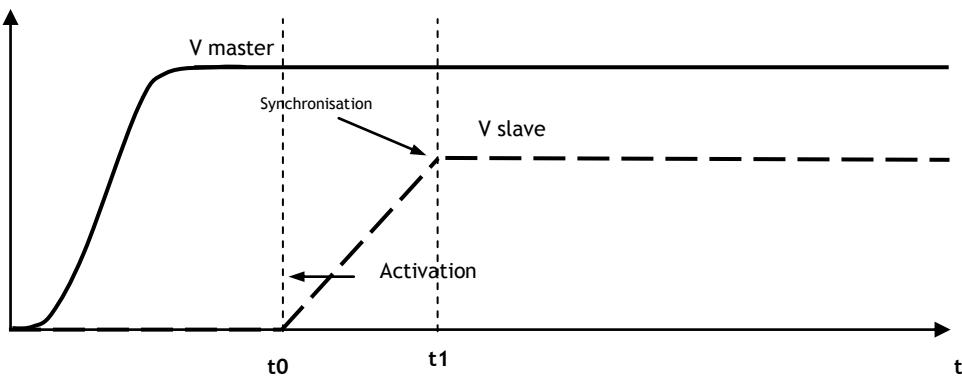
$$\text{PosSlave}_{tn+1} = \text{PosSlave}_{tn} + V_{slave_i} + A_{slave_i}$$

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed, using its deceleration ramp.

Mode_SPC

In this mode, the slave follows the position of the master proportionately to the value of the ratio (if the ratio = 1 the slave reproduces exactly the movement of the master); synchronisation depends on the dynamic characteristics of the slave axis and/or the Space parameter which defines the synchronisation distance.

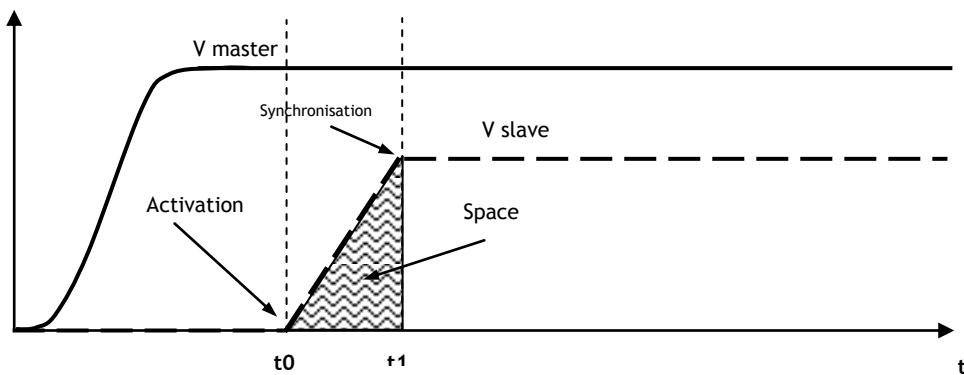
If the distance is 0, the slave will synchronise with the master based on its maximum acceleration and using linear ramps **only**.



If the distance is not 0, the slave axis will synchronise with the master axis based on an acceleration calculated as a function of the synchronisation distance and using linear ramps **only**. The acceleration value will be calculated again with each sampling step according to this formula:

$$A_{slave} = ((V_{master} * \text{FollowRate})^2 + V_{slave}^2) / 2 * \text{Space}$$

where the value of the distance is gradually reduced based on the distance covered during the synchronisation stage. No check is made on the ensuing acceleration value and therefore servo error messages may be generated the moment the acceleration exceeds the maximum value that the axis can withstand.



Once the synchronisation with the master axis has occurred, the slave will move according to the following formulas:

$$\text{PosSlave} = \text{PosSlave}_{t_1} + (\text{PosMaster} - \text{PosMaster}_{t_1}) * \text{FollowRate}$$

$$V_{\text{slave}} = V_{\text{master}} * \text{FollowRate}$$

The position, **PosSlave**, and the feed rate, **Vslave**, determined in this manner should be rated as “theoretical” values, since it is necessary to determine whether the values requested are compatible with the dynamic characteristics of the axis (Maximum feed rate and maximum acceleration). The moment the feed rate of the master varies, the slave will follow this variation according to its own acceleration value. If the feed rate requested for the slave exceeds the maximum value admissible for this axis, the system will reduce the feed rate accordingly. To this end, the two values with which to move the slave, **Vslave_i** and **Aslave_i** will be calculated instant by instant. The actual position of the slave axis will therefore be calculated on the basis of these values:

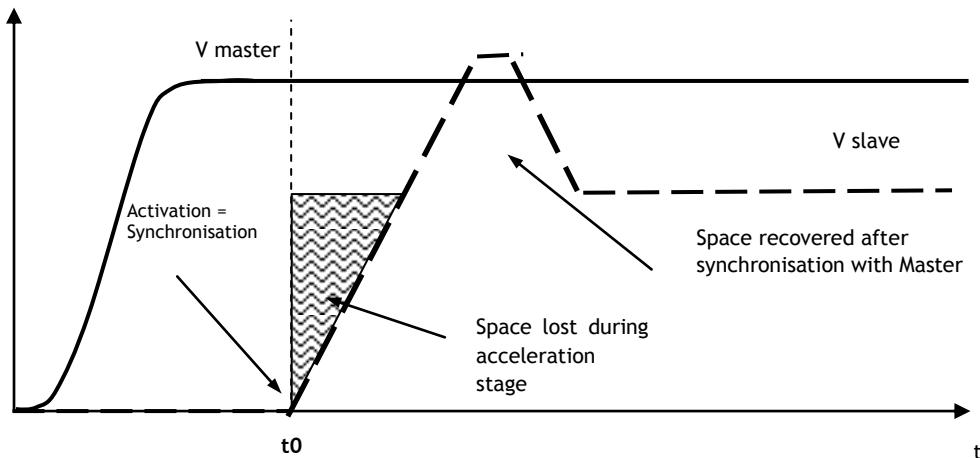
$$\text{PosSlave}_{t_{n+1}} = \text{PosSlave}_{t_n} + V_{\text{slave}} + A_{\text{slave}} * t$$

The difference between the actual and Theoretical position of the axis is taken up by the slave during its motion (even when the master has stopped moving) by moving, to the extent feasible, at a rate higher than Theoretical **Vslave**.

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed, using its deceleration ramp.

Mode_SPCVEL

In this mode, the slave follows the position and feed rate of the master proportionately to the value of



the ratio (if the ratio = 1, the slave reproduces exactly the movement of the master); synchronisation depends on the dynamic characteristics of the slave.

During the entire movement of the slave (i.e. both during and after the synchronisation stage), the motion of the axis is according to the following formulas (always using linear ramps):

$$\text{PosSlave} = \text{PosSlave}_{t_0} + (\text{PosMaster} - \text{PosMaster}_{t_0}) * \text{FollowRate}$$

$$V_{\text{slave}} = V_{\text{master}} * \text{FollowRate}$$

The position (**PosSlave**) and the feed rate (**Vslave**) determined in this manner should be rated as “theoretical” values, in that it is necessary to determine whether these requests are compatible with the dynamic characteristics of the axis (max admissible feed rate and max admissible acceleration).

The moment the feed rate of the master varies, the slave follows the variation according to its own acceleration value. If the feed rate requested of the slave is higher than its maximum admissible feed rate, the system reduces the feed rate requested accordingly. To this end, the two values with which the axis is to be moved (V_{slave_i} and A_{slave_i}) will be calculated instant by instant.

The actual position of the slave will therefore be calculated on the basis of these values:

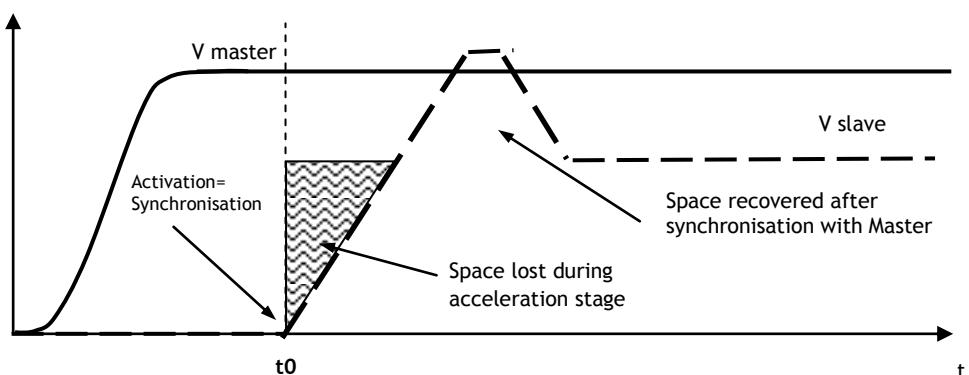
$$PosSlave_{tn+1} = PosSlave_{tn} + V_{slave_i} + A_{slave_i}$$

The difference between the actual and Theoretical position of the axis is taken up by the slave during its motion (even when the master has stopped moving) by moving, to the extent feasible, at a rate higher than Theoretical V_{slave} .

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed, using its deceleration ramp.

Mode_SPCVEL2

In this mode, the slave follows the position and feed rate of the master proportionately to the value of the ratio (if the ratio = 1, the slave reproduces exactly the movement of the master); synchronisation depends on the dynamic characteristics of the slave.



During the entire movement of the slave (i.e. both during and after the synchronisation stage), the motion of the axis is according to the following formulas (always using linear ramps):

$$PosSlave = PosSlave_{t0} + (PosMaster - PosMaster_{t0}) * FollowRate$$

$$V_{slave} = V_{master} * FollowRate$$

The position ($PosSlave$) and the feed rate (V_{slave}) determined in this manner should be rated as "theoretical" values, in that it is necessary to determine whether these requests are compatible with the dynamic characteristics of the axis (max admissible feed rate and max admissible acceleration). The moment the feed rate of the master varies, the slave follows the variation according to its own acceleration value. If the feed rate requested of the slave is higher than its maximum admissible feed rate, the system reduces the feed rate requested accordingly. To this end, the two values with which the axis is to be moved (V_{slave_i} and A_{slave_i}) will be calculated instant by instant. The actual position of the slave will therefore be calculated on the basis of these values:

$$PosSlave_{tn+1} = PosSlave_{tn} + V_{slave_i} + A_{slave_i}$$

The difference between the actual and Theoretical position of the axis is taken up by the slave during its motion (even when the master has stopped moving) by moving, to the extent feasible, at a rate higher than Theoretical **Vslave**.

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed, using its deceleration ramp. Unlike the other modes, once it has reached zero speed, the master axis will reposition itself at the point where it was the moment the deactivation request was made.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;  
Dummy   : dword;
```

```
Ret := MasterSlaveDefine (MODE_WAIT, Dummy, 1, MODE_COPY, 2.0, 0.0,  
                           1, 10, 11) ;
```

See also:

[MasterSlaveGO](#), [MasterSlaveParam](#), [MasterSlaveStatus](#), [MasterSlaveSTOP](#), [MasterSlaveUndefine](#)

2.19 MasterSlaveGO

The function MasterSlaveGO activates Master Slave following.

Syntax:

```
ret_code := MasterSlaveGO (ExecMode, ExecStatus, Process, SlaveName) ;
```

Input parameters

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

SlaveName (INT) Name/Id axis Slave

Possible overload:

MasterSlaveGO(INT,DWORD,INT,INT,...)

Execution mode:

Wait / NoWait

Use:

The following of the master by the slave is immediately activated. The following mode is defined by means of the MasterSlaveDefine function.

This command can be entered at any time (provided that the master/slave association has already been activated). The outcome of the command will tell whether the following command has been entered correctly and the request has been received by the axis movement function; the actual activation of the command must be monitored by the appropriate monitoring function.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example

```
Ret      : dword;
Dummy   : dword;
Ret := MasterSlaveGO (MODE_WAIT, Dummy, 1, 10, 11) ;
```

See also

[MasterSlaveDefine](#), [MasterSlaveParam](#), [MasterSlaveStatus](#), [MasterSlaveSTOP](#), [MasterSlaveUndefine](#)

2.20 MasterSlaveParam

The function `MasterSlaveParam` changes the Master Slave following ratio.

Syntax:

```
ret_code := MasterSlaveParam (ExecMode, ExecStatus, Process, FollRate, SlaveName) ;
```

Input parameters:

`ExecMode` (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

`ExecStatus` (DWORD) Command execution status

`Process` (INT) Process number [1..24]

`FollRate` (LREAL) Following ratio

`SlaveName` (INT) Name/Id axes Slave

Possible overload:

`MasterSlaveParam(INT,DWORD,INT,LREAL,INT,...)`

Execution mode:

Wait / NoWait

Use:

This instruction defines/changes the parameter that determines the master axis following rate on the part of the slave axes defined. Parameter `FollRate` defines the master following ratio specified for the slave(s). It must be viewed as a multiplication factor for the feed rate of the master or the distance covered by it. If the value of this ratio is 1.0, the motion of the master is reproduced exactly by the slave; if the ratio is smaller than 1.0, it is reduced, if the ratio is greater than 1.0 it is increased. This value can be preceded by a sign.

This command can be entered either when a slave is already following the master (in this case, the slave is released from the master and a new synchronisation stage gets underway based on the new following parameter) or when a slave is not in the following stage (then the following value to be adopted in the next movement is activated).

Return values

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;
Dummy   : dword;

Ret := MasterSlaveParam (MODE_WAIT, Dummy, 1, -1.3, 11) ;
```

See also:

`MasterSlaveGO`, `MasterSlaveDefine`, `MasterSlaveStatus`, `MasterSlaveDefine`, `MasterSlaveUndefine`

2.21 MasterSlaveStatus

Function MasterSlaveStatus can monitor the Master Slave following status.

Syntax:

```
ret_code := MasterSlaveStatus (Process, Status, SlaveName) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>SlaveName</i> (INT)	Name/Id axes Slave

Output parameters:

<i>Status</i> (XDA_Struct)	Structure to read the following status
----------------------------	--

Possible overload:

MasterSlaveStatus(INT,XDA_Struct,INT,...)

Execution mode:

Immediate

Use:

For all the required slaves, some data allowing to monitor the status of axes following are supplied within an XDA_Struct type structure. It can be defined up to 17 slave axes which must have been already associated with a master axis within that specified process. The given structure will be as follows:

Field	Type	Description
FollowingErr	Array of LREAL	Master/Slave distance
Status	Array of WORD	Following Status Word

Status	Mnemonic	Description
0x0001	MSL_UDASDA	UDA/SDA status enabled (with XDA at false)
0x0002	MSL_LINKED	Master axis linked
0x0004	MSL_RUNNING	Running enabled
0x0008	MSL_LINKING	Linking Phase
0x0010	MSL_RELEASEING	Releasing phase

Return values:

The following table lists all the possible *ret_code* variable values.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;
Status  : XDA_Struct;
Ret := MasterSlaveStatus (1, Status, 11) ;
```

See also:

[MasterSlaveGO](#), [MasterSlaveDefine](#), [MasterSlaveDefine](#), [MasterSlaveDefine](#), [MasterSlaveUndefine](#)

2.22 MasterSlaveSTOP

The function MasterSlaveSTOP deactivates the Master Slave following.

Syntax:

```
ret_code := MasterSlaveSTOP (ExecMode, ExecStatus, Process, SlaveName) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>SlaveName</i> (INT)	Name/Id axes Slave

Possible overload:

MasterSlaveSTOP(INT,DWORD,INT,INT,...)

Execution mode:

Wait / NoWait

Use:

The following of the master by the slave is deactivated immediately. The release mode is defined by means of the MasterSlaveDefine function.

The slave remains associated with the master, but does not follow it any more. Depending on the “mode_copy” parameter defined in the master/slave association command:

The slave switches immediately from the current feed rate to nil feed rate, **others** the slave comes to a halt according to its deceleration ramp.

The command can be entered at any time (provided that the master/slave association has already been activated).

The outcome of the command will tell whether the following deactivation command has been entered correctly and the request has been received by the axis movement function; **the actual activation of the command will have to be monitored by means of the appropriate monitoring function.**

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;
Dummy   : dword;
Ret := MasterSlaveSTOP (MODE_WAIT, Dummy, 1, 10, 11) ;
```

See also:

MasterSlaveGO, MasterSlaveParam, MasterSlaveStatus, MasterSlaveDefine, MasterSlaveUndefine

2.23 MasterSlaveUndefine

The function MasterSlaveUndefine removes the Master Slave association.

Syntax:

ret_code := MasterSlaveUndefine (*ExecMode*, *ExecStatus*, *Process*) ;

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode

Wait / NoWait

Use:

This instruction removes the master/slave association. This command can be accepted only if the process is in Idle.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example

```
Ret      : dword;
```

```
Dummy   : dword;
```

```
Ret := MasterSlaveUndefine (MODE_WAIT, Dummy, 1) ;
```

See also

MasterSlaveDefine, MasterSlaveGO, MasterSlaveParam, MasterSlaveStatus, MasterSlaveSTOP

2.24 PP_BlockSearch

The function PP_BlockSearch moves the part program to the requested N block number.

Syntax

```
ret_code := PP_BlockSearch (ExecMode, ExecStatus, Process, BlockNum, Direction) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>BlockNumber</i> (DINT)	N block number where positioning
<i>Direction</i> (BOOL)	Movement direction

Execution mode

Wait / NoWait

Use

The function moves the pointer to the next N block *BlockNumber* contained in the PP to be executed searching ahead and back from the current position, according to the *Direction* parameter that can have a value as SEARCH_AHEAD (false) for search ahead or SEARCH_BACK (true) for search back.

If a line placement is requested beyond the program limits, the pointer will be moved to the file start or end, according to the selected search direction.

Return values

The following table lists all the values the *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150025	Negative block number
0x00170005	Process number wrong or non-existent
0x0017005C	Part Program not enabled
0x00170058	Part program read error
0x00170050	End of the part program reached
0x00170052	Start of the part program reached

Example

```
Ret      : dword;
Dummy    : dword;
Ret := PP_BlockSearch (MODE_NOWAIT, Dummy, 1, 100, SEARCH_AHEAD) ;
```

See also

[PP_LineSearch](#), [PP_MarkerSearch](#), [PP_TextSearch](#)

2.25 PP_BlockNumberSearch

The function PP_BlockNumberSearch positions the pointer on the active Part Program of block number backwards or forwards.

Syntax:

```
ret_code := PP_BlockNumberSearch (ExecMode, ExecStatus, Process, BlockNumber, Direction) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>BlockNumber</i> (DINT)	Blocks number to jump
<i>Direction</i> (BOOL)	Movement direction

Execution mode:

Wait / NoWait

Use:

This function positions the pointer to the next PP block to be executed, "block number" blocks forwards or backwards with respect to the current position on the basis of the Direction parameter that can have SEARCH_AHEAD (false) value for search forwards or SEARCH_BACK (true) for search backwards .

If a move of a number of blocks greater than those left at the start or end of the Part Program is requested, the function indicates a beginning or end of file warning, according to the selected search direction.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Functions executed without errors
0x00150025	Negative block number
0x00170005	Process number wrong or not existent
0x0017005C	Part Program not active
0x00170058	Part program reading error
0x00170050	Part program end achieved
0x00170052	Part program start achieved

Example

```
Ret      : dword;
Dummy    : dword;
Ret := PP_BlockNumberSearch (MODE_NOWAIT, Dummy, 1, 100, true) ;
```

See also

[PP_BExecUntilBlockNumber](#), [PP_GetBlockNumber](#), [PP_TextSearch](#), [PP_Select](#), [PP.GetName](#)

2.26 PP_ExecUntilBlockNumber

Function PP_ExecUntilBlockNumber selects a part program block as end point.

Syntax:

```
ret_code := PP_ExecUntilBlockNumber (ExecMode, ExecStatus, Process, BlockNum) ;
```

Input Parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>BlockNum</i> (DINT)	Blocks number to jump

Execution mode:

Wait / NoWait

Use:

This function predisposes the part program active to end after the execution of the block marked with number *BlockNum*. The *BlockNum* is entered for the active program (main). The program must be started with the cycle start push button or with the equivalent PLC function.

If the block number specified in *BlockNum* is not present in the active program, the function does not return anything error (function executed correctly) and the part program will be entirely executed.

The command is cancelled when the part program positions itself on the indicated block, when the system is reset or when another program is selected.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150025	Negative block number
0x00170005	Process number wrong or not existent

Example

```
Ret      : dword;
Dummy    : dword;
Ret := PP_ExecUntilBlockNumbe (MODE_NOWAIT, Dummy, 1, 100) ;
```

See also

[PP_BlockNumberSearch](#), [PP_GetBlockNumber](#), [PP_TextSearch](#), [PP_Select](#), [PP_GetName](#)

2.27 PP_ExecUntil

The function PP_ExecUntil selects end of the part-program execution.

Syntax

```
ret_code := PP_ExecUntil (ExecMode, ExecStatus, Process, Marker, BlockNum, LineNum) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Marker</i> (STRING)	Marker where the part-program stops
<i>BlockNum</i> (DINT)	N block number where the part-program stops
<i>LineNum</i> (DINT)	Line number where the part-program stops
<i>BrkVal</i> (DINT)	Break value (break)

Execution mode

Wait / NoWait

Use

The function prepares the active part program to terminate after the execution of the line that meets the conditions specified in the call parameters;

The conditions for stopping the part-program are specified hierarchically by the parameters *Marker*, *BlockNum*, *BrkVal* and *LineNum*. If the *Marker* string is null, the marker stop condition is ignored, same if the *BlockNum* is negative (the condition of N block number is ignored). If *BrkVal* is negative, the break condition (break) is ignored as if *LineNum* was negative. For example, if all the parameters are enabled, the program will terminates the execution after finding the marker, the block number, the break condition (break) and after reaching and executing the line specified. The parameters are valid only for the active program (main).

The program execution must be launched with the cycle start button or with the equivalent PLC function. If the conditions specified are not present in the active program, the function does not return any error (function executed without errors) and the part-program will be fully executed. The command is reset when reaching the condition indicated or on reset or on selection of a different program.

Return values

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret      : dword;
Dummy    : dword;
Ret := PP_ExecUntil (MODE_NOWAIT, Dummy, 1, 'INIT', 100, -1, -1) ;
```

See also

PP_BlockSearch, PP_GetBlockNumber, PP_TextSearch, PP_GetMarker, PP_LineSerach, PP_MarkerSearch, PP_ExecUntilNext

2.28 PP_ExecUntilNext

The function PP_ExecUntilNext selects the part program execution stop and keeps the execution until the next condition.

Syntax

```
ret_code := PP_ExecUntilNext (ExecMode, ExecStatus, Process, Marker, BlockNum, LineNum,  
                  NextMode) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Marker</i> (STRING)	Marker where the part-program stops
<i>BlockNum</i> (DINT)	N block number where the part-program stops
<i>LineNum</i> (DINT)	Line number where the part-program stops
<i>BrkVal</i> (DINT)	Break value (break)
<i>NextMode</i> (INT)	Next condition where the part-program stops

Execution mode

Wait / NoWait

Use

The function prepares the active part-program to finish after the execution of the line that meets the conditions specified in the call parameters; reached this term continues until the condition specified by *NextMode*.

The conditions for stopping the part-program are specified hierarchically by the parameters *Marker*, *BlockNum*, *BrkVal* and *LineNum*. If the *Marker* string is null, the marker stop condition is ignored, same if the *BlockNum* is negative (the condition of N block number is ignored). If *BrkVal* is negative, the break condition (break) is ignored as if *LineNum* was negative. For example, if all the parameters are enabled, the program will terminate the execution after finding the marker, the block number, the break condition (break) and after reaching and executing the line specified.

Once the condition for stopping is found, the execution continues until an N number block is encountered , if condition *NextMode* = *NEXT_BLOCK* (0) ; or until it encounters a marker, if condition *NextMode* = *NEXT_MARKER* (1) ; or until the break value (break) reaches a value higher than the stop value if condition *NextMode* = *NEXT_BREAK* (2)

The parameters are valid only for the active program (main). The program execution must be launched using the cycle start button or by the equivalent PLC function. If the conditions specified are not found in the active program, the function does not return any error (function executed without errors) and the part-program will be fully executed. The command is reset when reaching the condition indicated or by reset or by selection of a different.

Return values

The following table lists all the values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret      : dword;
Dummy    : dword;
Ret := PP_ExecUntilNext(MODE_NOWAIT, Dummy, 1, 'INIT', -1, -1, -1, NEXT_BLOCK);
```

See also

[PP_BlockSearch](#), [PP_GetBlockNumber](#), [PP_TextSearch](#), [PP_GetMarker](#), [PP_LineSerach](#), [PP_MarkerSearch](#), [PP_ExecUntil](#)

2.29 PP_GetBlockNumber

Function PP_GetBlockNumber determines the active part program block sequence numbers.

Syntax:

```
ret_code := PP_GetBlockNumber (Process, Main, Level1, Level2, Level3, Level4) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameters:

<i>Main</i> (DINT)	Block number main
<i>Level1</i> (DINT)	Block number nesting level 1
<i>Level2</i> (DINT)	Block number nesting level 2
<i>Level3</i> (DINT)	Block number nesting level 3
<i>Level4</i> (DINT)	Block number nesting level 4

Execution mode:

Immediate

Use:

This function returns the active part program block sequence numbers of the main program and nested subroutines. If no block numbers are used in the active part program or in any of the subroutines, a 0 will be returned.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;
LevelMain : dint;
LevelSub1 : dint;
LevelSub2 : dint;
LevelSub3 : dint;
LevelSub4 : dint;

Ret := PP_GetBlockNumber (1, LevelMain, LevelSub1, LevelSub2, LevelSub3,
                         LevelSub4) ;
```

See also:

[PP_ExecUntilBlockNumber](#), [PP_BlockNumberSearch](#), [PP_TextSearch](#), [PP_Select](#), [PP.GetName](#)

2.30 PP_GetMarkers

The PP_GetMarkers function reads the active markers when a part program is running.

Syntax:

ret_code := PP_GetMarkers (*Process*, *Markers*) ;

Input parameter

<i>Process</i> (INT)	Process number [1..24]
<i>Mode</i> (INT)	Marker type required

Output parameter

<i>Markers</i> (MARKER_Struct)	Structure where markers are read
--------------------------------	----------------------------------

Execution mode

Immediate

Use

Within a MARKER_Struct structure type, it reads the active markers when a part program is running or the markers determined by the search of the “Closer point”. In the first case, Mode must be used as = MARKE_EXEC(0), in the second case Mode will be as = MARKER_NEAR (1). Markers are: line number, block number (defined by the ISO N code) and the marker (defined using the iso code \$”): each of them for the several part program execution levels at the moment of the call. In the structure there is also the present execution level of the part program and the value of the break variable.

The read structure will have the following information:

Field	Type	Description
ActLevel	INT	Execution level
BrkVal	DINT	Break value (break)
LineNum	Array of UDINT	Line number in execution
BlkNum	Array of UDINT	Block number in execution (N)
Marker	STRING Array	Active marker (\$”)
ProgName	STRING Array	Program name or Subroutine level

Return values

The following table lists all values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret : dword;
Info : MARKER_Struct;

Ret := PP_GetMarkers (1, Info) ;
```

See also: PP_GetBlockNumber

2.31 PP_MarkerSearch

The function PP_MarkerSearch moves the part program to the required marker.

Syntax

```
ret_code := PP_MarkerSearch (ExecMode, ExecStatus, Process, SearchMarker, Direction) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Marker</i> (STRING)	Marker where position
<i>Direction</i> (BOOL)	Movement direction

Execution mode

Wait / NoWait

Use

The function places the pointer at the next *Marker* contained in the PP to be executed searching backwards or forwards from the current position according to the *Direction* parameter that can have value SEARCH_AHEAD (false) for search forward or SEARCH_BACK (true) for search backward.

If a pointer at a line beyond the program limits is requested, the program will signal if the start or the end of the file is reached, depending on the search direction selected.

Return values

The following table lists all values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150025	Negative block number
0x00170005	Process number wrong or non-existent
0x0017005C	Part Program disabled
0x00170058	Part program read error
0x00170050	Part program end reached
0x00170052	Part program start reached

Example

```
Ret      : dword;
Dummy    : dword;
Ret := PP_MarkerSearch (MODE_NOWAIT, Dummy, 1, 'INIT', SERACH_AHEAD) ;
```

See also

[PP_LineSearch](#), [PP_BlockSearch](#), [PP_TextSearch](#)

2.32 PP_GetName

The function PP_GetName determines the name of the active part program.

Syntax:

```
ret_code := PP_GetName (Process, MainName, SubrName, Level) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameters:

MainName (STRING) Name of the active main part program

SubrName (STRING) Name of the active subroutine

Level (INT) Nesting level of the subroutine

Execution mode:

Immediate

Use:

The function returns the name of the active main part program, subroutine name in execution, if any and if present, and nesting level of the active part program.

If no part program is active (*Level* =0), the output parameters *MainName* and *SubrName* become insignificant.

If the main part program (*Level*=1) is in execution, its name is shown in the output parameter *MainName* while the *SubrName* parameter is insignificant.

If a subroutine (2<=Level<=5) is in execution, the *MainName* parameter will contain the name of the main part program while the *SubrName* parameter will contain the name of the subroutine in execution.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret      : dword;
MainName : string;
SubName  : string;
Level    : int;
Ret := PP_GetName (1, MainName, SubName, Level) ;
```

See also:

[PP_ExecUntilBlockNumber](#), [PP_BlockNumberSearch](#), [PP_TextSearch](#), [PP_Select](#), [PP_GetBlockNumber](#)

2.33 PP_LineSearch

The function PP_LineSearch places a pointer in the part program to the requested line.

Syntax

```
ret_code := PP_LineSearch (ExecMode, ExecStatus, Process, LineNumber, Mode) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>LineNumber</i> (DINT)	Line number where positioning
<i>Mode</i> (INT)	Positioning mode

Execution mode

Wait / NoWait

Use

The function places the pointer at the line *LineNumber* to be executed. The placement is made according to the mode defined by the *Mode* parameter: in case of MOVE_UP (0) it moves the pointer to the *LineNumber* line to be executed towards the program start, if MOVE_DOWN (1) it moves the pointer to the *LineNumber* line to be executed towards the program end, if MOVE_FIX (2) it moves the pointer to the *LineNumber* line.

If is required a placement to a line beyond the program limits, the program will signal if the start or the end of the file is reached, depending on the selected search direction.

Return value

The following table lists all the values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150025	Negative block number
0x00170005	Process number wrong or non-existent
0x0017005C	Part Program disabled
0x00170058	Part program read error
0x00170050	Part program end reached
0x00170052	Part program start reached

Example

```
Ret      : dword;
Dummy    : dword;
Ret := PP_LineSearch (MODE_NOWAIT, Dummy, 1, 100, MOVE_FIX) ;
```

See also

[PP_BlockSearch](#), [PP_MarkerSearch](#), [PP_TextSearch](#)

2.34 PP_Resume

Function PP_Resume resumes part program execution after MAS.

Syntax:

```
ret_code := PP_Resume (ExecMode, ExecStatus, Process) ;
```

Input parameters

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function is used to exit the MAS state of the NC which was forced by an M function (M06 tool change for example).

Return values:

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example

```
Ret      : dword;
Dummy    : dword;
Ret := PP_Resume (MODE_NOWAIT, Dummy, 1) ;
```

See also

[CycleStop](#), [CycleStart](#), [ReadProcessInfo](#)

2.35 PP_RunInfoGO

The function PP_RunInfoGO activates monitoring of a part program execution.

Syntax:

```
ret_code := PP_RunInfoGO (Process, Mode) ;
```

Input parameters:

Process (INT) Process number [1..24]
Mode (BOOL) Monitoring mode

Execution mode:

Immediate

Use:

Activates monitoring of Part Program active on process specified. The Mode parameter may also request the determination of the physical positions of the axes that are being moved in the presence of views and/or Tool Centre Point function. This parameter may be either NORM_INFO (false), which reads the programmed positions only, or WITH_ABS_INFO (true), which provides additional info on physical positions.

Return values:

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret : dword;  
Ret := PP_RunInfoGO (1, WITH_ABS_INFO) ;
```

See also:

[PP_RunInfoSTOP](#), [PP_RunInfoRead](#), [PP_RunInfoStatus](#)

2.36 PP_RunInfoRead

The function PP_RunInfoRead reads monitoring data of a part program execution.

Syntax:

```
ret_code := PP_RunInfoRead (Process, Info) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameters:

Info (PPRunInfo_Struct) Structure to read monitoring data

Execution mode:

Immediate

Use:

In a type PPRunInfo_Struct structure, this function reads the execution status of a part program. The monitoring activity must be activated beforehand with function PP_RunInfoGO.

The reading structure gives the following information:

Field	Type	Description
AxisId	Array of INT	Axes identifiers associated to the process
AxStatus	Array of WORD	Axes status
FinalWrk	Array of LREAL	Final position on current block
FinalWrkNxt	Array of LREAL	Final position on subsequent block
FinalAbs	Array of LREAL	Absolute final position on current block
FinalAbsNxt	Array of LREAL	Absolute final position on subsequent block
ProcStatus	WORD	Process status
NumBlok	DINT	Number of current block
NumBlokNxt	DINT	Number of subsequent block
TypeOfMove	INT	Movement type on current block
TypeOfMoveNxt	INT	Movement type on subsequent block
ProgFeed	LREAL	Programmed feed on current block (unit/min)
ProgFeedNxt	LREAL	Programmed feed on subsequent block (unit/min)
Radius	LREAL	Programmed radius on current block
RadiusNxt	LREAL	Programmed radius on subsequent block

AxStatus	Mnemonic	Description
0x0001	INFO_AXMOVED	Axis moved in current block
0x0002	INFO_AXMOVEDex	Axis moved indirectly in current block
0x0004	INFO_AXMOVED_ABS	Presence absolute position for current block
0x0010	INFO_AXnMOVED	Axis moved in subsequent block
0x0020	INFO_AXnMOVEDex	Axis moved indirectly in subsequent block
0x0040	INFO_AXnMOVED_ABS	Presence absolute position for subsequent block

ProcStatus	Mnemonic	Description
0x0001	INFO_PROCOK	Information for current block present
0x0010	INFO_PROCxOK	Information for subsequent block present
0x1000	INFO_exTENDED	Physical positions request activated
0x2000	INFO_UNCHANGED	Information not changed vs. previous reading
0x4000	INFO_RUNNING	Active monitoring
0x8000	INFO_PROCon	Process OK

TypeOfMove	Mnemonic	Description
1	INFO_LINEAR	Linear motion
2	INFO_CIRCLE	Circular motion
6	INFO_HSM	Polynomial motion
16	INFO_HOMING	Zero setting cycle
17	INFO_MANUAL	Motion in manual
32	INFO_PROBING	Probing cycle
48	INFO_THREAD	Thread cycle
64	INFO_FIXCYC	Standard fixed cycle
65	INFO_TAPPING	Tapping cycle without transducer
66	INFO_TAPPING_TRASD	Tapping cycle with transducer
67	INFO_BORING	Boring cycle

Return values:

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example:

```
Ret : dword;
Info : PPRunInfo_Struct;

Ret := PP_RunInfoRead (1, Info) ;
```

See also:

[PP_RunInfoSTOP](#), [PP_RunInfoGO](#), [PP_RunInfoStatus](#)

2.37 PP_RunInfoStatus

Function PP_RunInfoStatus returns a part program execution monitoring status

Syntax:

```
ret_code := PP_RunInfoStatus (Process, Status) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameters:

Status (BOOL) Monitoring status

Execution mode:

Immediate

Use:

This function determines whether the monitoring activity (activated with PP_RunInfoGO) is active on the parameter specified. If its *Status* is true, the monitoring activity is taking place, and conversely, if it is false, it is not.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example:

```
Ret : dword;  
        
Sts : BOOL ;  
        
Ret := PP_RunInfoStatus (1, Sts) ;
```

See also:

[PP_RunInfoSTOP](#), [PP_RunInfoRead](#), [PP_RunInfoGO](#)

2.38 PP_RunInfoSTOP

The function PP_RunInfoSTOP disables monitoring of a part program execution.

Syntax:

```
ret_code := PP_RunInfoSTOP (Process) ;
```

Input parameters:

Process (INT) Process number [1..24]

Execution mode:

Immediate

Use:

Disables monitoring of a part program execution on the specified process.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example:

```
Ret : dword;  
Ret := PP_RunInfoSTOP (1) ;
```

See also:

[PP_RunInfoGO](#), [PP_RunInfoRead](#), [PP_RunInfoStatus](#)

2.39 PP_Select

The function PP_Select activates a part program.

Syntax:

```
ret_code := PP_Select (ExecMode, ExecStatus, Process, Mode, PPName, ErroStr) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Mode</i> (INT)	Mode of activation
<i>PPName</i> (STRING)	Part program name

Output parameters:

<i>ErroStr</i> (STRING)	Error string to be returned
-------------------------	-----------------------------

Execution mode:

Wait / NoWait

Use:

This function is used to select or de-select a part program for execution. The activity can be done on the basis of *Mode* parameter.

Mode	Mnemonic	Description
0	RELEASE_PROGRAM	Active part program release
1	SELECT_PROGRAM	Part program selection
2	SELECT_DREEP_FEED	Activates execution in passing modality
3	SELECT_PROGRAM_DRV	Select a part program with logic name (drive)
4	SELECT_PROGRAM_WOA	Select a part program without analysing it
5	SELECT_PROGRAM_DRV_WOA	Select a part program with a logic name without analysing it
6	SWAP_PROGRAM	Swap between the selected program and a preloaded one
7	SWAP_PROGRAM_DRV	Swap between the selected program and a preloaded one with logical name
32	PRELOAD_PROGRAM	Preloads the Part Program
33	PRELOAD_PROGRAM_DRV	Preloads the Part Program with logical name
34	UNLOAD_PROGRAM	Unload the preloaded part program
35	UNLOAD_PROGRAM_DRV	Unload the preloaded part program with logical name
36	UNLOAD_ALL	Unloads all the preloaded part programs

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x00170043	Logic name (drive) not found
0x00170042	Name too long
0x0017003B	Access error to part program
0x00170059	Part program not found
0x0017005F	Directory part program not found

Example:

```
Ret      : dword;
Dummy    : dword;
ErroStr  : STRING ;
Ret := PP_Select (MODE_WAIT, Dummy, 1, SELECT_PROGRAM, 'MAIN_PROG.txt',
                  ErroStr) ;
```

See also

[PP_ExecUntilBlockNumber](#), [PP_GetBlockNumber](#), [PP_TextSearch](#), [PP_BlockNumberSearch](#), [PP.GetName](#)

2.40 PP_TextSearch

The function PP_TextSearch finds a string inside the part program active.

Syntax:

```
ret_code := PP_TextSearch (ExecMode, ExecStatus, Process, SearchString, Direction) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>SearchString</i> (STRING)	String to be found
<i>Direction</i> (BOOL)	Search direction

Execution mode

Wait / NoWait

Use:

This function searches for the string specified inside *SearchString* parameter in the selected part program on the basis of *Direction* parameter which may contain SEARCH_AHEAD (false) for search ahead or SEARCH_BACK (true) for search back.

At the next Part Program block to be executed, the pointer will be positioned on the block where the string has been found.

If the string is not found, the function returns the beginning or end of file, according to the selected search direction.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x0017005C	Part Program not active
0x00170058	Part program reading error
0x00170050	Part program end achieved
0x00170052	Part program start achieved

Example:

```
Ret      : dword;
Dummy    : dword;

Ret := PP_TextSearch (MODE_NOWAIT, Dummy, 1, 'G0Y0', true) ;
```

See also:

[PP_ExecUntilBlockNumber](#), [PP_GetBlockNumber](#), [PP_BlockNumberSearch](#), [PP_Select](#), [PP.GetName](#)

2.41 PP_LoadXmlMap

PP_LoadXmlMap function enables a new XML remapping file.

Syntax

```
ret_code := PP_LoadXmlMap (ExecMode, ExecStatus, Process, MapName) ;
```

Input parameters

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
ExecStatus (DWORD) Command execution status
Process (INT) Process number [1..24]
MapName (STRING) Remapping file PathName

Execution mode

Wait / NoWait

Use

This function is used to upload a new remapping file that executes part program instructions in XML format.

Return values

The following table lists all the possible values of *ret_code* variable:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent
0x001700B1	Mapping file not found, release failed
0x001700B2	Syntax error in the mapping file

Example

```
Ret      : dword;
Dummy   : dword;

Ret := PP_LoadXmlMap (MODE_WAIT, Dummy, 1, '\User\Macro\ 'XMLMAP.txt') ;
```

2.42 ProtectAreaDefine

Function ProtectAreaDefine defines a safety area.

Syntax:

```
ret_code := ProtectAreaDefine (ExecMode, ExecStatus, Process, Areald, PathName, Type) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>execStatus (DWORD)</i>	Command execution status
<i>Process (INT)</i>	Process number [1..24]
<i>Areald (INT)</i>	Area identifier [1..4]
<i>PathName (STRING)</i>	PartProgram containing the area definition
<i>Type (INT)</i>	Specifies if internal/external area

Execution mode:

Wait / NoWait

Use:

This function uploads in the memory a protected area defined within the Part Program *PathName*. The Part Program syntax defining the area presents the following limitations:

- X and Y are namely the abscissa and the ordinate axes even if they are not defined within the process. Safety area definition refers to an internal and absolute Cartesian system, therefore during the activation, the relation between “absolute” and “real” reference system has to be specified.
- The first block of the Part Program defining the area identifies its starting point. This block contains the point abscissa and ordinate and doesn't contain the G2 or G3 codes.
- The only G codes allowed are G0, G1, G2 and G3
- There are no triliteral or logic functions
- Default moves (safety area sides) are linear

The safety area shows the following features:

- Being traced by 10 linear or circular elements (maximum)
- Being closed: start and final points correspond. If the Part Program description doesn't refer to a closed profile, CN will apply a linear closing move. The added element cannot exceed the maximum number allowed.
- Being convex

The tool head will always stay inside or outside the area (Internal or external type) according to the *Type* value.

Type	Mnemonic	Description
0	PROTECT_INTERNAL	Internal type protected area
1	PROTECT_EXTERNAL	External type protected area

ProtectAreaEnable2D or *ProtectAreaEnable2D_2* functions activate the area making it operative.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret : dword;
Ret := ProtectAreaDefine(MODE_NOWAIT, ULO, 1, 1, 'Area1',
                         PROTECT_EXTERNAL);
```

See also:

[ProtectAreaDisable](#), [ProtectAreaEnable2D](#), [ProtectAreaEnable2D_2](#)

2.43 ProtectAreaDisable

Function ProtectAreaDisable disables a safety area.

Syntax:

```
ret_code := ProtectAreaDisable (ExecMode, ExecStatus, Process, Areald) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Areald</i> (INT)	Area identifier [1..4]

Execution mode:

Wait / NoWait

Use:

Disables a protected area identified by *Areald*.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret : dword;  
Ret := ProtectAreaDisable (MODE_NOWAIT, ULO, 1, 1) ;
```

See also:

[ProtectAreaDefine](#), [ProtectAreaEnable2D](#), [ProtectAreaEnable2D_2](#)

2.44 ProtectAreaEnable2D

Function ProtectAreaEnable2D enables a 2D safety area.

Syntax:

```
ret_code := ProtectAreaEnable2D (ExecMode, ExecStatus, Process, AreaId, XAxisId, XAxisOffs,  
                  YAxisId, YAxisOffs, Origin, Mode) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>AreaId</i> (INT)	Area identifier [1..4]
<i>XAxisId</i> (INT)	First axis identifier [1..64]
<i>XAxisOffs</i> (LREAL)	Offset on the first axis
<i>YAxisId</i> (INT)	Second axis area identifier [1..64]
<i>YAxisOffs</i> (LREAL)	Offset on the second axis
<i>Origin</i> (INT)	Origin number on the axes
<i>Mode</i> (INT)	Activation/deactivation tool compensation

Execution mode:

Wait / NoWait

Use:

This function enables a protected area previously defined by the ProtectAreaDefine function. XAxisId and YAxisId parameters specify the plane referring to the defined area. XAxisOffs and YAxisOffs parameters relocate the protected area from the origin of reference, specified by Origin.

Origin	Mnemonic	Description
-1	WITH_CURRENT_ORIGIN	Active on current origin
n		Origin number to use

Mode parameter allows the cutter diameter compensation in the area to be set. In that circumstance, the working area is limited according to the tool dimensions, otherwise is limited according to the tool centre.

Mode	Mnemonic	Description
0	WITHOUT_COMPENSATION	Without considering the tool compensation
1	WITH_COMPENSATION	Considering the tool compensation



“RESET” doesn’t disable the protected area. Call the *ProtectAreaDisable* function or reboot the system to disable protected area.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret : dword;  
(* Enables safety area 1 on axes 10 and 11 with a 40mm offset on the first. In  
this area is activated the cutter diameter compensation *)  
Ret := ProtectAreaEnable2D (MODE_NOWAIT, ULO, 1, 1, 10, 40.0, 11, 0.0,  
                           WITH_CURRENT_ORIGIN, WITH_COMPENSATION) ;
```

See also:

[ProtectAreaDefine](#), [ProtectAreaDisable](#), [ProtectAreaEnable2D_2](#)

2.45 ProtectAreaEnable2D_2

Function ProtectAreaEnable2D_2 enables a 2D and ½ safety area (with vertical axis).

Syntax:

```
ret_code := ProtectAreaEnable2D_2 (ExecMode, ExecStatus, Process, Areald, XAxisId, XAxisOffs,
                                  YAxisId, YAxisOffs, ZAxisId, ZAxisOffs ZlowLimit, ZUpLimit,
                                  Origin, Mode) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Areald</i> (INT)	Area identifier [1..4]
<i>XAxisId</i> (INT)	First axis identifier [1..64]
<i>XAxisOffs</i> (LREAL)	Offset on the first axis
<i>YAxisId</i> (INT)	Second axis area identifier [1..64]
<i>YAxisOffs</i> (LREAL)	Offset on the second axis
<i>ZAxisId</i> (INT)	Vertical axis area id [1..64]
<i>ZAxisOffs</i> (LREAL)	Offset on the vertical axis
<i>ZLowLimit</i> (LREAL)	Vertical axis lower limit
<i>ZUpLimit</i> (LREAL)	Vertical axis upper limit
<i>Origin</i> (INT)	Origin number on the axes
<i>Mode</i> (INT)	Activation/deactivation tool compensation

Execution mode:

Wait / NoWait

Use:

This function enables a protected area previously defined by the *ProtectAreaDefine* function. *XAxisId* and *YAxisId* parameters specify the plane to which the defined area refers. *ZAxisId* indicates the axis to use for the vertical extension of the area, while *ZLowLimit* and *ZUpLimit* identify the upper/lower limits of the extension.

XAxisOffs, *YAxisOffs* and *ZAxisOffs* relocate the protected volume from the area origin of reference specified by *Origin* parameter.

Origin	Mnemonic	Description
-1	WITH_CURRENT_ORIGIN	Active on current origin
n		Origin number to use

Mode parameter allows the cutter diameter compensation of the area. In this case, the area defines the working area not according to the tool dimensions, but considering only its centre.

Mode	Mnemonic	Description
0	WITHOUT_COMPENSATION	Without considering the tool compensation
1	WITH_COMPENSATION	Considering the tool compensation



“RESET” doesn’t disable the protected area. Call the *ProtectAreaDisable* function or reboot the system to disable the area.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret : dword;  
(* Activates the protected area 1 on axes with ID 10, 11 and 12 offsetting the  
first of 40mm. Axis 12 is limited between -100.0 and 80.0. On this area is  
activated the cutter diameter compensation *)  
Ret := ProtectAreaEnable2D_2 (MODE_NOWAIT, ULO, 1, 1, 10, 40.0, 11, 0.0,  
12, 0.0, -100.0, 80.0, WITH_CURRENT_ORIGIN,  
WITH_COMPENSATION) ;
```

See also:

[ProtectAreaDefine](#), [ProtectAreaDisable](#), [ProtectAreaEnable2D](#)

2.46 ReadProcessInfo

Function ReadProcessInfo reads the status of a process.

Syntax:

```
ret_code := ReadProcessInfo (Process, Info) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameters:

Info (PROCESSInfo_Struct) Structure containing process status

Execution mode

Immediate

Use

This function reads the execution status of a process within a type PROCESSInfo_Struct data structure input as parameter to the function. This structure contains the following data:

Field	Type	Description
ProcSts	WORD	Process status
ProcCW	WORD	Process Control Word
ProcMode	WORD	Active mode on the process
ProcType	WORD	Process type
ProcErr	DWORD	Error present on the process
Paramacro	WORD	Paramacro G active
FixCycleSts	WORD	Active fixed cycle status
VirtuSts	WORD	Active virtualizations
Gcode_00_15	WORD	Bitmap presence G0..G15
Gcode_16_31	WORD	Bitmap presence G16..G31
Gcode_32_47	WORD	Bitmap presence G32..G47
Gcode_48_63	WORD	Bitmap presence G48..G63
Gcode_64_79	WORD	Bitmap presence G64..G79
Gcode_80_95	WORD	Bitmap presence G80..G95
Gcode_96_99	WORD	Bitmap presence G96..G99
StoredBlk	DWORD	Number of continuous mode movements analysed
ExecudBlk	DWORD	Number of continuous mode movements executed
Odh	WORD	Execution status (debug)
RapidOv	WORD	Value of Rapid Override present %
ManualOv	WORD	Value of Manual Override present %
FeedOv	WORD	Value of Feed Override present %
SpindleOv	WORD	Value of Spindle Override present %
IntpSts	WORD	Interpolator execution status
IntpSub	WORD	Execution sub-status (for the INT_ST_STOP stage)
FholdSts	WORD	Feedhold status
JogInc	LREAL	Value of Jog Incremental present

ActVel	LREAL	Current feed rate (unit/min)
ActVell	LREAL	Current feed rate (unit/clock)
DistToGo	LREAL	Distance To Go to end of block being executed
DistCov	LREAL	Distance covered in the block of execution
Angle	LREAL	Current angle for circular motion (in radians)
DistToGoHold	LREAL	Distance To Go to end of block in hold
DistCovHold	LREAL	Distance covered in block on hold
AngleHold	LREAL	Current angle for circular movement (in radian) in hold
TappingError	LREAL	Tapping/Threading error (for monitoring)
TappingPos	LREAL	Tapping/Threading axis position
FeedValue	LREAL	Programmed feed
SelectedAxis	Array INT	Axes array selected for manual motions

With the OPENcontrol 2.1. release, also the following information will be available:

Field	Type	Description
AcceValue	LREAL	Programmed acceleration
DeceValue	LREAL	Programmed deceleration
JrkAValue	LREAL	Jerk in programmed acceleration
JrkDValue	LREAL	Jerk in programmed deceleration
TminAValue	LREAL	Minimum ramp time in programmed acceleration.
TminDValue	LREAL	Minimum ramp time in programmed deceleration.
DistCovGlb	LREAL	Distance covered from the profile start

From the OPENcontrol 3.2 release, the following information will be also available:

Field	Type	Description
MovStat	WORD	Movement status
MovSubs	WORD	Movement sub-status
LineMain	DINT	Line execution on Main
LineLevl	DINT	Line execution at PartProgram level
Level	INT	Execution level
RealtPPIInfo	Array LREAL	PartProgram realtime information

For the fields described before:

ProcSts	Mnemonic	Description
0x0001	_SW_IDLE	Process in status of IDLE
0x0002	_SW_CYCLE	Active cycle
0x0004	_SW_HOLDA	Process in status of HOLD
0x0008	_SW_RUNH	Process in status to enter in HOLD
0x0010	_SW_HRUN	Process in status to exit from HOLD
0x0020	_SW_ERRO	Process in status of ERROR
0x0040	_SW_RUNPP	Program execution in HOLD or MAS
0x0080	_SW_RESET	Process in status of RESET
0x0100	_SW_EMERG	Process in status of EMERGENCY
0x0200	_SW_WAIT	Process in status of WAIT
0x0400	_SW_INPUT	Process in status of INPUT
0x0800	_SW_SUB_MBR	Process in sub status Multi Block Retrace
0x2000	_SW_SUB_MAS	Process in sub status of MAS

0x8000	_SW_FHOLD	Process in status of FeedHold
--------	-----------	-------------------------------

ProcCW	Mnemonic	Description
0x0001	_SW_UNIT	Process measurement unit1 inch 0 mm
0x0002	_SW_MIDIPARAM	Paramacro executed in MDI
0x0400	_SW_RCMDONE	Search in memory executed
0x0800	_SW_AUX	Presence of auxiliary functions after RCM
0x1000	_SW_RCM	Search in memory active (RCM)
0x2000	_SW_DRY	DryRun enabled
0x4000	_SW_EOB	End of Block function enabled
0x8000	_SW_FRB	Feed ByPass enabled

ProcMode	Mnemonic	Description
0x0001	_SW_MDI	Executive mode MDI
0x0002	_SW_AUTO	Executive mode AUTO
0x0004	_SW_STEP	Executive mode STEP by STEP
0x0008	_SW_MANU	Executive mode MANUAL
0x0010	_SW_MANJ	Executive mode MANUAL JOG
0x0020	_SW_RET PROF	Executive mode of JOG RETURN
0x0040	_SW_HOME	Executive mode HOMING
0x0080	_SW_HPG	Executive mode HandWheel

xCycleSts	Mnemonic	Description
0x0001	_SW_INVER	Spindle reversal request
0x0002	_SW_STOPR	Spindle stop request
0x0100	_SW_TRAPID	Rapid approach in probe mode
0x0004	_SW_TEACH0	Function 0 teach pendant button
0x0008	_SW_TEACH1	Function 1 teach pendant button
0x0010	_SW_TEACH2	Function 2 teach pendant button
0x0020	_SW_TEACH3	Function 3 teach pendant button
0x0040	_SW_TEACH4	Function 4 teach pendant button
0x0080	_SW_TEACH5	Function 5 teach pendant button
0x0200	_SW_TEACH6	Function 6 teach pendant button
0x0400	_SW_TEACH7	Function 7 teach pendant button
0x0800	_SW_TEACH8	Function 8 teach pendant button
0x1000	_SW_TEACH9	Function 9 teach pendant button

VirtuSts	Mnemonic	Description
0x0001	_SW_XDA	XDA enabled
0x0002	_SW_UA	UDA enabled
0x0004	_SW_SDA	SDA enabled

dh	Mnemonic	Description
0x0001	ODH_DEFG	Geometric entity deficit
0x0002	ODH_SKIP	Entity skipped due to high speed
0x0004	ODH_OVERA	Over acceleration

0x0008	ODH_DEFID	Deficit dynamic entities
--------	-----------	--------------------------

IntpSts	Mnemonic	Description
0	INT_ST_FREE	Missing interpolator
1	INT_ST_IDLE	Interpolator disabled
2	INT_ST_RUN	Interpolation running
3	INT_ST_STOP	Interpolation stopped
4	INT_ST_HOLD	Interpolator in Hold
5	INT_ST_ERR	Interpolator in error
6	INT_ST_EMG	Interpolator in emergency
7	INT_ST_RESE	Interpolator in reset

IntpSub	Mnemonic	Description
0	INT_ST_STOH	Braking interpolator to Hold
1	INT_ST_STOR	Braking interpolator to Reset
2	INT_ST_STOE	Braking interpolator to Emergency
3	INT_ST_STOX	Braking interpolator to Error
4	INT_ST_STOQ	Braking interpolator for Stop request

FholdSts	Mnemonic	Description
0x8000	FHOLD_ON	FeedHold status on the interpolator
0x4000	FHOLD_ONEMOV	FeedHold status at movement end
0x2000	FHOLD_INTERP	FeedHold status internal
0x1000	FHOLD_WAIT	FeedHold status for Dwell
0x0001	FAST_STOP	Fast stop required
0x0002	FAST_MANLOP	Fast stop required on manual limits
0x0004	FAST_MANHALT	Halt request on manual limits
0x8000	FHOLD_ON	FeedHold status on the interpolator

MovStat	Mnemonic	Description
0	MOV_ST_NULL	Non-existent movement
1	MOV_ST_STRT	Movement in the start phase
2	MOV_ST_IDLE	Movement in idle
3	MOV_ST_RUN	Movement running
4	MOV_ST_HOLD	Movement in Hold
5	MOV_ST_STOP	Movement stopped
6	MOV_ST_EMER	Movement in emergency
7	MOV_ST_TOLL	Movement in axes tolerance phase

MovSubs	Mnemonic	Description
0	MOV_ST_INIT	Movement initialisation (MOV_ST_RUN)
1	MOV_ST_NORM	Movement running (MOV_ST_RUN)
2	MOV_ST_STOH	Movement in Hold stop (MOV_ST_STOP)
3	MOV_ST_STOR	Movement in Reset stop (MOV_ST_STOP)
4	MOV_ST_STOE	Movement in Emergency stop (MOV_ST_STOP)
5	MOV_ST_STOX	Movement stopped (MOV_ST_STOP)

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160132	Call not valid for process 0
0x0016012C	Process non-existent

Example:

```
Ret      : dword;
ProInfo  : PROCESSInfo_Struct;

Ret := ReadProcessInfo (1, ProInfo) ;
```

See also

CycleStop, CycleStart, HolON, HoldOFF, EndReset, Reset, PP_Resume, ExecMDIBlock, SetProcessMode, SetFeedOverride, SetManualOverride, SetRapidOverride, SetSpindleOverride, GetJogIncrement, SetJogIncrement, EnterFeedHold, ExitFeedHold, EndOfBlockEnable, EndOfBlockDisable, FeedBypassON, FeedBypassOFF, SetSearchInMemory

2.47 SetProcessMode

The function SetProcessMode sets mode of operation for the process.

Syntax:

ret_code := SetProcesMode (*ExecMode*,*ExecStatus*, *Process*, *ProcessMode*) ;

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

ProcessMode (INT) RCM execution mode

Execution mode:

Wait / NoWait

Use:

Select the process operative mode according to the *ProcessMode* parameter that is:

ProcSts	Mnemonic	Description
1	MODE_MDI	Request mode MDI
2	MODE_AUTO	Request mode AUTO
3	MODE_BLKBLK	Request mode STEP (block by block)
4	MODE_CONTJOG	Request mode MANUAL
5	MODE_INCRJOG	Request mode JOG INCREMENTAL
6	MODE_JOGRETURN	Request mode Return on Profile
7	MODE_HOMING	Request mode HOMING
8	MODE_HPG	Request mode Hand wheel

Return values:

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x001700A0	Mode wrong

Example

```
Ret      : dword;
Dummy   : dword;
Ret := SetProcessMode (MODE_WAIT, Dummy, 1, MODE_MDI) ;
```

See also

[ReadProcessInfo](#)

2.48 Reset

The Reset function activates the reset of a process.

Syntax:

```
ret_code := Reset (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

A system reset causes:

- › all axes motion to stop
- › the part program buffer to be cleared
- › the part program pointer to be set to the first block of the active program
- › the G92 work register pre-sets to be cancelled
- › all origin shifts to be cleared
- › the cycle stop button to be released
- › the power turn on G-codes to be activated
- › certain three-letter codes to be reset

All other actions required for a reset must be started in the WinPLUS program. These actions can be different according to the type of applications. Some examples of what may be necessary to execute from WinPLUS:

- › stop the spindle and the coolant
- › abort all point-to-point-axis motion
- › reset specific M codes to their default state
- › activate a default screen display
- › initialize certain machine related conditions and variables

The sequence of **RESET** usage in a PLC should be:

- › call **RESET**
- › waiting enter in reset by the process
- › execute the action required for **RESET**
- › call **ENDRESET**

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example :

```
Ret      : dword;
Dummy    : dword;

Ret := Reset (MODE_NOWAIT, Dummy, 1) ;
```

See also:

[CycleStop](#), [CycleStart](#), [HolON](#), [HoldOFF](#), [EndReset](#), [ReadProcessInfo](#)

2.49 EndReset

Function EndReset notifies end of reset from PLC.

Syntax:

```
ret_code := EndReset (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This system function call must be used to tell the system that PLC has finished all the action required for a RESET.

The system will not go into IDLE state before it receives this call from the PLC. If this function is not called after reset, the system will "hang" in the reset status.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example

```
Ret      : dword;
Dummy   : dword;
Ret := EndReset (MODE_NOWAIT, Dummy, 1) ;
```

See also

[CycleStop](#), [CycleStart](#), [HoldON](#), [HoldOFF](#), [Reset](#), [ReadProcessInfo](#)

2.50 SelectAxis

Function SelectAxis is used to select one or more axes for manual operations.

Syntax:

```
ret_code := SelectAxis (ExecMode, ExecStatus, Process, AxisId) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

AxisId (INT) Id axes to be selected

Possible overload:

SelectAxis(INT,DWORD,INT,INT,...)

Execution mode:

Wait / NoWait

Use:

This function is used to select one or more axes for manual operations. The selected axes must belong to specified process.

Up to 12 axes can be selected. The AxisId parameter can be duplicated for each one of the axes to be selected.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150020	Axis not associated with process
0x00150022	Axis not definite
0x00150023	Axes identifier not expected
0x00150027	Too many axes selected
0x00170005	Process number wrong or not existent
0X001700A0	Too many axes in comparison with axes belonged to the process

Example:

```
Ret      : dword;
Dummy   : dword;
Ret := SelectAxis (MODE_WAIT, Dummy, 1, 1, 2, 3) ;
```

See also:

SelectAxisAndOverride, SelectAxisManualOverride

2.51 SelectAxisAndOverride

The function `SelectAxisAndOverride` is used to select one or more axes to be moved manually and the associated Override.

Syntax:

```
ret_code := SelectAxisAndOverride (ExecMode, ExecStatus, Process, AxisOverride) ;
```

Input parameters:

`ExecMode` (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

`ExecStatus` (DWORD) Command execution status

`Process` (INT) Process number [1..24]

`AxisOverride` (AXISAndOverride_Struct) Axes to be selected

Possible overload:

`SelectAxisAndOverride(INT,DWORD,INT,AXISAndOverride_Struct,...)`

Execution mode:

Wait / NoWait

Use:

This function is used to select one or more axes to be moved manually and to define for each of them the feed rate override percentage. Only the axes associated with the active process can be selected. Up to 12 axes can be selected. The value of `Override` is expressed in per cent * 100 i.e. 10% = 1000 and it must be defined inside the `AXISAndOverride_Struct` structure together with the axis Id to be selected. Since the `Override` value is defined by a signed INT field, the value of the sign will determine the direction of the movement; if +, it will move positive, if - it will move negative.

The axis speed (and consequently its direction) will result according to the following formula:

$$V_a = V_c * |S_f| * S_a$$

V_a = axis speed

V_c = configured speed in ODM in the “manual feed” field

S_f = percentage value of the feed manual Override or value forced through `SetManualOverride` function. The configured direction (Jog+ o Jog-) is not taken into consideration, only the absolute value.

S_a = set value in the function `Override` field.



The final direction of the axis movement is determined by the sign of the **V_a** variable; since only the **S_a** value is signed, the **V_a** sign is influenced only by **S_a**.

The moment only one axis is selected, the first time its sign changes the **S_f** variable is again considered as signed (so it influence the direction of the axis movement); so only the first movement is executed in the direction defined by **S_a**; when the **S_f** sign is changed, this also will be taken into consideration for the direction of the movement.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150020	Axis not associated with process
0x00150022	Axis not defined
0x00150023	Axes identifier not expected
0x00150027	Too many axes selected
0x00170005	Process number wrong or non-existent
0X001700A0	Too many axes in comparison with axes belonging to the process

Example

```

Ret      : dword;
Dummy    : dword;
First    : AXISAndOverride_Struct;
Second   : AXISAndOverride_Struct;

First.AxisId    := 1 ;
First.Override  := 4000 ;
Second.AxisId   := 2 ;
Second.Override := -4000 ;

Ret := SelectAxisAndOverride (MODE_WAIT, Dummy, 1, First, Second) ;

```

See also

[SelectAxis](#), [SelectAxisManualOverride](#), [SetManualOverride](#)

2.52 SetAxisManualOverride

The function SetAxisManualOverride selects Override for the axes moved manually.

Syntax:

```
ret_code := SetAxisManualOverride (ExecMode, ExecMode, Process, AxisOverride) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

AxisOverride (AXISAndOverride_Struct) Axes to be selected

Possible overload:

SetAxisManualOverride(INT,DWORD,INT,AXISAndOverride_Struct,...)

Execution mode:

Wait / NoWait

Use:

This function is used to set the feed rate override for the axes moved manually. Only the axes associated with the defined process can be selected. Up to 12 axes can be selected. The value of *feed* is expressed in per cent * 100 i.e. to have 10% =1000 and it must be defined inside the AXISAndOverride_Struct structure together with the axis Id to be selected. Since the *Override* value is defined by a signed INT field, the value of the sign will determine the direction of the movement; if +, it will move positive, if - it will move in negative.

The axis speed (and consequently its direction) will result according to the following formula:

$$V_a = V_c * |S_f| * S_a$$

V_a = Axis speed

V_c = configured speed in ODM in the field “manual feed”

S_f = percentage value of the feed manual Override or value forced through SetManualOverride function. The configured direction (Jog+ o Jog-) is not taken into consideration, only the absolute value.

S_a = set value in the function *Override* field.



The final direction of the axis movement is determined by the sign of the **V_a** variable; since only the **S_a** value is marked with a sign, the **V_a** sign is influenced only by **S_a**.

The moment one axis is selected, the first time its sign changes the **S_f** variable is again considered as signed (so it influence the direction of the axis movement); thus only the first movement is executed in the direction defined by **S_a**; when the **S_f** sign is changed, this also will be taken into consideration for the direction of the movement.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150020	Axis not associated with process
0x00150022	Axis not defined
0x00150023	Axes identifier not expected
0x00150027	Too many axes selected
0x00170005	Process number wrong or non-existent
0X001700A0	Too many axes in comparison with axes belonging to the process

Example:

```

Ret      : dword;
Dummy    : dword;
First    : AXISAndOverride_Struct;
Second   : AXISAndOverride_Struct;

First.AxisId    := 1 ;
First.Override  := 4000 ;
Second.AxisId   := 2 ;
Second.Override := -4000 ;

Ret := SelectAxisManualOverride (MODE_WAIT, Dummy, 1, First, Second) ;

```

See also:

[SelectAxis](#), [SelectAxisAndOverride](#), [SetManualOverride](#)

2.53 EndOfBlockdDisable

Function EndOfBlockdDisable disables the filter request on block execution end.

Syntax:

```
ret_code := EndOfBlockdDisable (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function restores the system's normal functioning. Consensus to analyse the following block will not be requested to the interface at the end-of-block (part program block pre-calculation enabled).

Return values:

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret      : dword;
Dummy   : dword;
Ret := EndOfBlockDisable (MODE_WAIT, Dummy , 1) ;
```

See also

[EndOfBlockEnable](#), [ReadProcessInfo](#)

2.54 EndOfBlockEnable

The function EndOfBlockdEnable enables the filter request on block execution end.

Syntax:

```
ret_code := EndOfBlockEnable (ExecMode, ExecStatus, Process) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function disables the part program block pre-calculation. Before analysing another block, the system requires the consent to the function of the EOB part program filter. A "NACK" force the system in "MAS+IDLE" state.

If the nEOB routine is enabled, it will also be activated when the "#" synchronization character is placed in front of a part program block.

Return values

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret      : dword;
Dummy   : dword;
Ret := EndOfBlockEnable (MODE_WAIT, Dummy , 1) ;
```

See also

[EndOfBlockDisable](#), [ReadProcessInfo](#)

2.55 SetFilterConsole

Function SetFilterConsole specifies whether or not the PLC is informed of console events generated in the system.

Syntax:

```
ret_code := SetFilterConsole (Process, Event, Enable, DefRisplnt, DefRispExt) ;
```

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>Event</i> (DINT)	Console event whose PLC must be informed
<i>Enable</i> (BOOL)	Enable/Disable functionality
<i>DefRisplnt</i> (BOOL)	Default reply for requests by operator console
<i>DefRispExt</i> (BOOL)	Default reply for requests by external environment
<i>PlcFiltEnab</i> (BOOL)	Enable filter also by PLC request

Execution mode

Immediate

Use:

Enables or disables the filter management function on console requests. If the filter is enabled, the PLC will be informed of every event generated; for each of these events, the PLC will send a signal to the system concerning the acquisition of the event (see AckStrobeConsole). If the filter is disabled, the system will not inform the PLC of console events, but will use the default response configured, if any; if no response has been configured, the default response is ACK (OK, console event acknowledge).



Once the console event filter management function has been enabled, the system will freeze if the PLC does not reply to a console event notice with function AckStrobeConsole.

The *Event* variable is used to enable the filter on a specific console event.

This parameter may be:

Event	Value	Description
f_CYCLE_ON	0x00000001	Cycle Start notice
f_CYCLE_OFF	0x00000002	Cycle Stop notice
f_RESET	0x00000004	Reset notice
f_HOLD_ON	0x00000008	Hold request notice
f_HOLD_OFF	0x00000010	Exit from Hold request notice
f_FEEDMANUAL_OV	0x00000020	Override on manual feed notice
f_FEED_OV	0x00000040	Override on feed notice
f_SPEED_OV	0x00000080	Override on speed notice
f_SETMODE	0x00000100	Mode change request notice
f_RAPID_OV	0x00000200	Override on rapid feed notice
f_SELAXI	0x00000400	Selection axes request notice

The *Enable* parameter is used to determine whether or not to enable the communication of an event.

Enable	Value	Description
	true	Enables the acknowledge request
	false	Disables the acknowledge request

In this case, if the *PlcFiltEnab* is a TRUE, also the events generated by the requests of a PLC task will be reported; on contrary, the requests of a PLC task will not be filtered.

Parameter *DefRispInt* supplies the default response to be used for requests coming from the operator console when the filter for that particular event has been disabled.

DefRispInt	Value	Description
	true	ACK signal is given (command acknowledged)
	false	NACK signal is given (command refused)

Parameter *DefRispEst* supplies the default response to be used for requests coming from the external environments as WinNBI when the filter for that particular event has been disabled.

DefRispEst	Value	Description
	true	ACK signal is given (command acknowledged)
	false	NACK signal is given (command refused)

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Event parameter wrong
0x00150020	Process parameter wrong (outside range)
0x00150021	Process not existent

Example:

```
Ret : dword;
Ret := SetFilterConsole ( 1, f_CYCLE_ON, true, true, false) ;
Ret := SetFilterConsole ( 1, f_CYCLE_OFF, false, true, true) ;
```

See also:

[AckStrobeConsole](#), [GetFilterConsole](#), [ClearStrobeConsole](#), [ReadFilterConsole](#).

2.56 ReadFilterConsole

Function ReadFilterConsole reads the data associated with a console event.

Syntax:

```
ret_code := ReadFilterConsole (Process, ConsoleData) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameters:

ConsoleData (ANY) Structure to read data

Possible overload:

ReadFilterConsole (INT,CYCON_FilterStruct)
 ReadFilterConsole (INT,CYCOFF_FilterStruct)
 ReadFilterConsole (INT,RESET_FilterStruct)
 ReadFilterConsole (INT,HOLDON_FilterStruct)
 ReadFilterConsole (INT,HOLDOFF_FilterStruct)
 ReadFilterConsole (INT,FEEDMAN_FilterStruct)
 ReadFilterConsole (INT,FEEDRATE_FilterStruct)
 ReadFilterConsole (INT,SPEEDRATE_FilterStruct)
 ReadFilterConsole (INT,FEEDRAP_FilterStruct)
 ReadFilterConsole (INT,SETMODE_FilterStruct)
 ReadFilterConsole (INT,SELAXI_FilterStruct)

Execution mode:

Immediate

Use:

When the system has notified a console event (whose notification has been requested with the SetFilterConsole function), it becomes possible to read the data associated with it.

For each console event there is an ad hoc structure suitable to contain the relative data, and hence the structure implicitly defines the event associated with it; for this reason, the read function only has to supply the *ConsoleData* structure, and it need not supply the event. Inside all these structures, there is a field, *Ambient*, which identifies the sender of the command. It may assume the following values:

Ambient (HEX)	Mnemonic	Description
0x3050	CONSOLE_AMBIENT	Console ambient
0x3052		External ambient (WinNbi)



To be informed of a console event and be able to read the data, it is necessary to enable the relative management beforehand by means of the SetFilterConsole function. Reading the data without having enabled the console filter management function may yield non-significant values.

Through the GetStrobeConsole function it is possible to determine the console event notified and present in the system.

If function GetStrobeConsole returns the presence of event **f_CYCLE_ON**, the relative data must be read in a type **CYCON_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting the Cycle Start

Or **CYCONex_FilterStruct** can be used for filter on Cycle Start command (extended)

Field	Type	Description
Ambient	WORD	Ambient requiring Cycle Start
Mode	INT	Cycle Start execution mode

Mode	Mnemonic	Description
0	NORM_CYCLE	Simple Cycle
4	POSJOG_CYCLE	Cycle in manual with JOG +
5	NEGJOG_CYCLE	Cycle in manual with JOG -

If function GetStrobeConsole returns the presence of event **f_CYCLE_OFF** the relative data must be read in a type **CYCOFF_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting the Cycle Stop

If function GetStrobeConsole returns the presence of event **f_RESET** the relative data must be read in a type **RESET_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting the Reset

If function GetStrobeConsole returns the presence of event **f_HOLD_ON** the relative data must be read in a type **HOLDON_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting enter in hold

If function GetStrobeConsole returns the presence of event **f_HOLD_OFF** the relative data must be read in a type **HOLDOFF_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting exit from hold

If function GetStrobeConsole returns the presence of event **f_FEEDMANUAL_OV** the relative data must be read in a type **FEEDMAN_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting variation feed override for manual movement
ValuePerc	INT	Override value expressed in 0.01 %
Mode	INT	Reserved
Direction	INT	Direction

Direction	Mnemonic	Description
0	POSITIVE_DIR	Movement in positive direction
1	NEGATIVE_DIR	Movement in negative direction

If function GetStrobeConsole returns the presence of event **f_FEED_OV** the relative data must be read in a type **FEEDRATE_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting feed override variation
ValuePerc	INT	Override value expressed in 0.01 %
Mode	INT	Reserved

If function GetStrobeConsole returns the presence of event **f_SPEED_OV** the relative data must be read in a type **SPEEDRATE_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting speed override variation
ValuePerc	INT	Override value expressed in 0.01 %
Mode	INT	Reserved

If function GetStrobeConsole returns the presence of event **f_RAPID_OV** the relative data must be read in a type **FEEDRAP_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting rapid override variation
ValuePerc	INT	Override value expressed in 0.01 %
Mode	INT	Reserved

If function GetStrobeConsole returns the presence of event **f_SETMODE** the relative data must be read in a type **SETMODE_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting operative mode variation
SelMode	INT	New operative mode

SelMode	Mnemonic	Description
1	MODE_MDI	Execution mode in MDI
2	MODE_AUTO	Execution mode in automatic
3	MODE_BLKBLK	Execution mode in semiautomatic
4	MODE_CONTJOG	Execution mode in continuous manual
5	MODE_JOGINCR	Execution mode in incremental manual
6	MODE_JOGRETURN	Return mode on profile
7	MODE_HOMING	Homing mode
8	MODE_HPG	Hand wheel mode

If function GetStrobeConsole returns the presence of event **f_SELAXI** the relative data must be read in a type **SELAXI_FilterStruct** structure organised as follows:

Field	Type	Description
Ambient	WORD	Ambient requesting axes selection
Type	INT	Type of axes selection
AxisId	Array of INT	Selected axes identifiers
AxisPerc	Array of INT	Override value expressed in 0.01 % for axis

Type	Mnemonic	Description
1	SELAXI_AX	Simple axes selection (only Id axes)
2	SELAXI_AXFED	Axes selection and feed for axes
3	SELAXI_FED	Variation % override for axes already selected

If GetStrobeConsole function returns the **f_SETJOG** event, the data in a **JOGVAL_FilterStruct** structure will be read as follow:

Field	Type	Description
Ambient	WORD	Ambient requiring axes selection
Value	LREAL	Jog value

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150020	Process parameter wrong (outside range)
0x00150021	Process not existent

Example:

```

Ret      : dword;
FdRate   : FEEDRATE_FilterStruct ;
FdSetmode: SETMODE_FilterStruct;
Event    : dint ;

Ret     := GetStrobeConsole(1, Event);
CASE Event OF
  f_FEED_OV :
    Ret := ReadFilterConsole (1, FdRate );
  f_SETMODE :
    Ret := ReadFilterConsole (1, FdSetmode);
END_CASE;

```

See also:

[SetFilterConsole](#), [AckStrobeConsole](#), [ClearStrobeConsole](#), [GetStrobeConsole](#)

2.57 GetStrobeConsole

Function GetStrobeConsole determines the console event present in the system.

Syntax:

```
ret_code := GetStrobeConsole (Process, Event) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameters:

Event (DINT) Console event present in the system

Execution mode:

Immediate

Use

If the console event filter management function has been enabled (see SetFilterConsole), every console event generated in the system will be communicated to the PLC. This function determines whether the console event is present in the system.

The console event concerned will be notified via the *Event* variable which is an input parameter to this function. It will contain the following values:

Event	Value	Description
	0x00000000	No event present
f_CYCLE_ON	0x00000001	Cycle Start event
f_CYCLE_OFF	0x00000002	Cycle Stop event
f_RESET	0x00000004	Reset event
f_HOLD_ON	0x00000008	Hold request event
f_HOLD_OFF	0x00000010	Hold exit request event
f_FEEDMANUAL_OV	0x00000020	Override event on manual feed
f_FEED_OV	0x00000040	Override event on feed
f_SPEED_OV	0x00000080	Override event on speed
f_SETMODE	0x00000100	Request event of Mode change
f_RAPID_OV	0x00000200	Override event on rapid feed
f_SELAXI	0x00000400	Request event of axes selection

Return values

The following table gives the values that may be assumed by variable *ret_code*

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150020	Process parameter wrong (outside range)
0x00150021	Process not existent

Example:

```
Ret    : dword;
Event  : dword;
Ret := GetStrobeConsole (1, Event) ;
```

See also:

[SetFilterConsole](#), [ClearStrobeConsole](#), [AckStrobeConsole](#), [ReadFilterConsole](#)

2.58 AckStrobeConsole

The AckStrobeConsole function informs the system that a Console event has been acquired by the PLC and communicates the result thereof.

Syntax:

```
ret_code := AckStrobeConsole (Process, AckNack, ConsoleData) ;
```

Input parameters

Process (INT) Process number [1..24]

AckNack (BOOL) Acknowledge given/not given to system for event notified

ConsoleData (ANY) Data for Acknowledge given/not given to system for event notified.

Possible overload

AckStrobeConsole (INT,BOOL,CYCON_FilterStruct)

AckStrobeConsole (INT,BOOL,CYCOFF_FilterStruct)

AckStrobeConsole (INT,BOOL,RESET_FilterStruct)

AckStrobeConsole (INT,BOOL,HOLDON_FilterStruct)

AckStrobeConsole (INT,BOOL,HOLDOFF_FilterStruct)

AckStrobeConsole (INT,BOOL,FEEDMAN_FilterStruct)

AckStrobeConsole (INT,BOOL,FEEDRATE_FilterStruct)

AckStrobeConsole (INT,BOOL,SPEEDRATE_FilterStruct)

AckStrobeConsole (INT,BOOL,FEEDRAP_FilterStruct)

AckStrobeConsole (INT,BOOL,SETMODE_FilterStruct)

AckStrobeConsole (INT,BOOL,SELAXI_FilterStruct)

Execution mode

Immediate

Use:

If the management of Console filters has been enabled (see SetFilterConsole), any console request generated in the system is communicated to the PLC. The moment it has acquired the communication (see GetStrobeConsole), the PLC must inform the system of the acquisition by giving out an acknowledge through the variable *AckNack*.

In the case of Console filters, the PLC may modify the values communicated by the system and make other values operational; for example, if the system transmits a 50% feed override request, the PLC may respond with a true ACK (i.e., request accepted) and at the same time change the 50% value to a different value, e.g., 30%. To do so, it must read the data communicated by the system within ad hoc structures (see function ReadFilterConsole), change the data contained therein, and notify the structure concerned with function AckStrobeConsole.

For each console event there is an ad hoc structure suitable to contain the relative data, and hence the structure implicitly defines the event associated with it; for this reason, only the structure needs to be supplied to the acknowledge function, and the event needs not. Inside all these structures, there is a field, *Ambient*, which identifies the sender of the command.

It may assume the following values:

Ambient (HEX)	Mnemonic	Description
0x3050	CONSOLE_AMBIENT	Console ambient
0x3052		External ambient (WinNbi)

Upon receiving this acknowledge, the system is able to notify further subsequent console requests.



The system will freeze if the PLC does not reply to a console event with the AckStrobeConsole function. Needless to say, this is so provided that the PLC has activated the filter management function on Console events.

The AckNack parameter defines the reply mode and will be

AckNack	Value	Description
ACK	true	Request accepted by the PLC
NACK	false	Request not accepted by the PLC

If the PLC does not acknowledge the request, the system will generate an error message.

CYCON_FilterStruct for filter on Cycle Start command

Field	Type	Description
Ambient	WORD	Ambient requesting the Cycle Start

Mode	Mnemonic	Description
0	NORM_CYCLE	Simple Cycle
4	POSJOG_CYCLE	Cycle in manual with JOG +
5	NEGJOG_CYCLE	Cycle in manual with JOG -

CYCONex_FilterStruct for filter on Cycle Start command:

Field	Type	Description
Ambient	WORD	Ambient requiring the Cycle Start
Mode	INT	Cycle Start execution mode

Mode	Mnemonic	Description
0	NORM_CYCLE	Simple Cycle
4	POSJOG_CYCLE	Cycle in manual with JOG +
5	NEGJOG_CYCLE	Cycle in manual with JOG -

CYCOFF_FilterStruct for filter on Cycle Stop command

Field	Type	Description
Ambient	WORD	Ambient requesting the Cycle Stop

RESET_FilterStruct for filter on Reset command

Field	Type	Description
Ambient	WORD	Ambient requesting the Reset

HOLDON_FilterStruct for filter on entry to Hold command

Field	Type	Description
Ambient	WORD	Ambient requesting to entry in hold

HOLDOFF_FilterStruct for filter on exit to Hold command

Field	Type	Description
Ambient	WORD	Ambient requesting to come out of hold

FEEDMAN_FilterStruct for filter on feed override variation in manual mode command

Field	Type	Description
Ambient	WORD	Ambient requesting feed override variation in manual mode command
ValuePerc	INT	Override value expressed in 0.01 %
Mode	INT	Reserved
Direction	INT	Direction

Direction	Mnemonic	Description
0	POSITIVE_DIR	Movement in positive direction
1	NEGATIVE_DIR	Movement in negative direction

FEEDRATE_FilterStruct for filter on feed override variation command

Field	Type	Description
Ambient	WORD	Ambient requesting feed override variation
ValuePerc	INT	Override value expressed in 0.01%
Direction	INT	Direction : If 0 positive direction, 1 negative

SPEEDRATE_FilterStruct for filter on speed override variation command

Field	Type	Description
Ambient	WORD	Ambient requesting speed override variation
ValuePerc	INT	Override value expressed in 0.01%
Mode	INT	Reserved

FEEDRAP_FilterStruct for filter on rapid override variation command

Field	Type	Description
Ambient	WORD	Ambient requesting rapid override variation
ValuePerc	INT	Override value expressed in 0.01%
Mode	INT	Reserved

SETMODE_FilterStruct for filter on operative mode variation command

Field	Type	Description
Ambient	WORD	Ambient requesting operative mode variation
SelMode	INT	New operating mode

SelMode	Mnemonic	Description
1	MODE_MDI	MDI execution mode
2	MODE_AUTO	Automatic execution mode
3	MODE_BLKBLK	Semiautomatic execution mode
4	MODE_CONTJOG	Continuous jog mode execution
5	MODE_JOGINCR	Incremental manual execution
6	MODE_JOGRETURN	Jog return mode
7	MODE_HOMING	Homing mode
8	MODE_HPG	Hand wheel mode

SELAXI_FilterStruct for filter on axes selection command

Field	Type	Description
Ambient	WORD	Ambient requesting axes selection
Type	INT	Axes selection type
AxisId	Array of INT	Identifier selected axes
AxisPerc	Array of INT	Override value expressed in 0.01 % for the axis

Type	Mnemonic	Description
1	SELAXI_AX	Simple axis selection (only Id axes)
2	SELAXI_AXFED	Axis and axis feed selection
3	SELAXI_FED	% override variation for axes already selected

Return values

The following table gives the possible values of variable *ret_code*:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Event parameter wrong
0x00150020	Process parameter wrong (outside range)
0x00150021	Process not existent

Example

```
Ret : dword;
FdRate : FEEDRATE_FilterStruct ;
Ret := AckStrobeConsole (1, ACK, FdRate) ;
```

See also

[SetFilterConsole](#), [GetStrobeConsole](#), [ClearStrobeConsole](#), [ReadFilterConsole](#).

2.59 AxisGet

AxisGet function allows a process to get/to release axes resources.

Syntax

```
ret_code := AxisGet (ExecMode, ExecStatus, Process, OrigMode, AxisId, AxisName,  
                  ToolOfsLenNum) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>OrigMode</i> (INT)	Origins management mode for the acquisition stage
<i>AxisId</i> (INT)	Axes Identifier [1..64]
<i>AxisName</i> (STRING[1])	Axis name
<i>ToolOfsLenNum</i> (INT)	Length offset index to apply on the axis [0..5]

Possible overloads

AxisGet (INT,DWORD,INT,INT,INT,STRING[1],INT[,INT,STRING(1),INT]...)

Execution mode

Wait / NoWait

Use

The function moves one or more axes from one process to the other, defining also offsets/axes relation. This function can be called only when the process is in IDLE, MBR and MAS status.

When using this function, all the axes used by the process have to be displayed, even if they already belong to the process. If an axis is not specified, it is released to be available for other processes (up to 12 axes can be specified). In order to release all process axes, the function has to be recalled specifying only one axis with *AxisId* 0 and *AxisName* ‘ ‘ (space).

Process axes viewing order is the one displayed by the function; the function can be used to change the axes viewing order. Rest instruction DOES NOT restore the changes due to the AxisGet function use. The control has to be rebooted in order to restore the axes configuration displayed in AMP.

During tool change operations (especially during offset enabling command) the axes to which associate the length offsets have to be already acquired by the process.

In some cases AxisGet function cannot be used:

- ▷ When tool radius offset mode is enabled (G41-G42)
- ▷ With an active canned cycle (G81-89)

In HOLD status

- ▷ In IDLE-MAS status
- ▷ With tool length offset mode enabled (h or T) and if leaving an axis having length offset mode enabled or when acquiring a new axis where the tool offset will be applied.
- ▷ When releasing axes shared with the logic

INFORMATION INITIALIZED AFTER THE FUNCTION EXECUTION

Interpolation plane

- ▷ Interpolation plane will be made by a couple among the three axes programmed according to the active plane change “G” --> G17, G18, G19.
- ▷ If axes defined are lower than 3, the first and second axis will be forced as abscissa and ordinate.
- ▷ G16 has to be programmed for the requested plane for a proper functioning.

Origins

OrigMode	Mnemonic	Description
0	GTA_ResetOrig	Active origins are reset
1	GTA_KeepOrigByID	Origins will be maintained on axes reprogrammed with same ID and order, even if they now have a different name.
2	GTA_KeepOrig	Origins will be maintained on all programmed axes even with different IDs.

Mirror

- ▷ Mirror will be reset on all axes.

Scale factor.

- ▷ Scale factor will be reset on all axes.

"ROT" rotation.

- ▷ Rotation enabled by "ROT" part program will be reset.

Operational limits.

- ▷ All axes will have operational limit values and the configuration.

Axis selection for manual motions.

- ▷ After the function execution, the selection of the first axis for manual motions, will be in viewing order.

Length offsets.

- ▷ Default associations defined in AMP will be lost. They will be enabled again after the control turning on.

Information kept after the execution

- ▷ Axis locked
- ▷ Incremental and temporary origins.

Return values

The following table lists all the possible values of *ret_code*

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret : dword;
(* Process gets the axis Id1 and gives it the name X not *)
(* assigning any length offset *)
Ret := AxisGet (MODE_WAIT, Status, 0, 1, 'X', 0);

(* Process releases all the axes *)
Ret := AxisGet (MODE_WAIT, Status, 0, 0, ' ', 0);
```

2.60 GetStrobePProgram

GetStrobePProgram function determines the PartProgram event of the system.

Syntax

ret_code := GetStrobePProgram (*Process*, *Event*) ;

Input parameters

Process (INT) Process number [1..24]

Output parameters

Event (DINT) PartProgram event in the system

Execution mode

Immediate

Use

If filters management is enabled in PartProgram events (see SetFilterPProgram), each PartProgram event created by the system will be indicated to the PLC. This function allows determination of the PartProgram event in the system.

The PartProgram event will be indicated int the *Event* variable defined as the function parameter. The values will be as follow:

Event	Value	Description
	0x00000000	No events
f_M_FUNCTION	0x00000001	M function event
f_S_FUNCTION	0x00000002	S function event
f_T_FUNCTION	0x00000004	T function event
f_CONSENT_TO_MOVE	0x00000008	Movement consent event
f_END_OF_MOVE	0x00000010	End of motion event
f_PSEUDO_AX_FUNCTION	0x00000020	Pseudo-axes event
f_EOB_FUNCTION	0x00000040	End Of Block event
f_RQP_FUNCTION	0x00000080	RQP function event
f_RQT_FUNCTION	0x00000100	RQT function event
f_TOU_FUNCTION	0x00000200	TOU function event
f_PROBE_FUNCTION	0x00000400	Probe feedback event
f_SHARE_FUNCTION	0x00000800	SHARE function event
f_SPG_FUNCTION	0x00001000	SPG function event
f_GTA_FUNCTION	0x00002000	GTA function event
f_TCP_FUNCTION	0x00004000	TCP function event
f_DUAL_FUNCTION	0x00008000	Dual axes function event

Return values:

The following table lists all the possible values of *ret_code*

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150020	Wrong Process Parameter (out of range)
0x00150021	Non-existent process

Example

```
Ret    : dword;
Event  : dword;
Ret := GetStrobePProgram (1, Event) ;
```

See also

[SetFilterPProgram](#), [ClearStrobePProgram](#), [AckStrobePProgram](#), [ReadFilterPProgram](#)

2.61 ClearStrobeConsole

Function ClearStrobeConsole resets (for the PLC only) the presence of a console event.

Syntax:

```
ret_code := ClearStrobeConsole (Process, Event) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>Event</i> (DINT)	Console event to be deleted

Execution mode:

Immediate

Use:

This function deletes a console event notified to the PLC. The moment a console event is generated, the system keeps it present and available for the PLC until the PLC acknowledges its acquisition through the AckStrobeConsole function. At this point, the system is ready to generate further console requests.

If an event is no longer to be notifiable (i.e., the GetStrobeConsole function will not notify any console event to the PLC) and at the same time freezes the console event management, use this function.

The *Event* parameter identifies the console event to be reset. Its values may be:

Event	Value	Description
f_CYCLE_ON	0x00000001	For Cycle Start event
f_CYCLE_OFF	0x00000002	For Cycle Stop event
f_RESET	0x00000004	For Reset event
f_HOLD_ON	0x00000008	For Hold request event
f_HOLD_OFF	0x00000010	For exit from Hold event
f_FEEDMANUAL_OV	0x00000020	For override on manual feed event
f_FEED_OV	0x00000040	For override on feed event
f_SPEED_OV	0x00000080	For override on speed event
f_SETMODE	0x00000100	For Mode change request event
f_RAPID_OV	0x00000200	For override on rapid feed event
f_SELAXI	0x00000400	For request axes selection event



The system will freeze if the PLC does not reply to a console event with the AckStrobeConsole function. This assumes that the PLC has activated the filter management function on Console events.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Event parameter wrong
0x00150020	Process parameter wrong (outside range)
0x00150021	Process non-existent

Example:

```
Ret : dword;  
Ret := ClearStrobeConsole (1, f_SELAXI) ;
```

See also:

[SetFilterConsole](#), [GetStrobeConsole](#), [AckStrobeConsole](#), [ReadFilterConsole](#)

2.62 SetFilterPProgram

Function SetFilterPProgram specifies whether or not the PLC is informed of PartProgram events generated in the system.

Syntax:

ret_code := SetFilterPProgram (*Process*, *Event*, *Enable*, *DefRisp*) ;

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>Event</i> (DINT)	PartProgram event whose PLC must be informed
<i>Enable</i> (BOOL)	Enable/Disable the functionality
<i>DefRisp</i> (BOOL)	Default reply
<i>PlcFiltEnab</i> (BOOL)	Enable filter also by PLC request

Execution mode:

Immediate

Use:

Enables or disables the filter management function on part program requests. If the filter is enabled, the PLC will be informed of every event generated; for each of these events, the PLC will send a signal to the system concerning the acquisition of the event (see AckStrobePProgram). If the filter is disabled, the system will not inform the PLC of part program events, but will use the configured default response, if any; if no response has been configured, the default response is ACK (acknowledge console event).



Once the part program event filter management function has been enabled, the system will freeze if the PLC does not reply to a part program event notice with theAckStrobePProgram function.

The *Event* variable is used to enable the filter on a specific part program event.

This parameter may be:

Event	Value	Description
f_M_FUNCTION	0x00000001	Notice for function M
f_S_FUNCTION	0x00000002	Notice for function S
f_T_FUNCTION	0x00000004	Notice for function T
f_CONSENT_TO_MOVE	0x00000008	Notice for movement consent
f_END_OF_MOVE	0x00000010	Notice for end movement
f_PSEUDO_AX_FUNCTION	0x00000020	Notice for pseudo axes
f_EOB_FUNCTION	0x00000040	Notice for End Of Block
f_RQP_FUNCTION	0x00000080	Notice for function RQP
f_RQT_FUNCTION	0x00000100	Notice for function RQT
f_TOU_FUNCTION	0x00000200	Notice for function TOU
f_PROBE_FUNCTION	0x00000400	Notice for probing result
f_GTS_FUNCTION	0x00000800	Notice for function GTS
f_SPG_FUNCTION	0x00001000	Notice for function SPG
f_GTA_FUNCTION	0x00002000	Notice for function GTA
f_TCP_FUNCTION	0x00004000	Notice for function TCP
f_DUAL_FUNCTION	0x00008000	Notice for dual axes function

The *Enable* parameter is used to enable or disable the event signal.

Enable	Value	Description
	TRUE	Enables the acknowledge request
	FALSE	Disables the acknowledge request

In this case, if the *PlcFiltEnab* is TRUE, the events generated by requests of a PLC task will be reported; otherwise, the requests generated by the PLC will not be filtered.

Parameter *DefRisp* supplies the default response when the filter for that particular event has been disabled.

DefRisp	Value	Description
	TRUE	ACK signal is given (command acknowledged)
	FALSE	NACK signal is given (command refused)

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Event parameter wrong
0x00150020	Process parameter wrong (outside range)
0x00150021	Process not existent

Example:

```
Ret : dword;
Ret := SetFilterPProgram ( 1, f_T_FUNCTION, true, true) ;
Ret := SetFilterPProgram ( 1, f_S_FUNCTION, false, true) ;
```

See also:

[AckStrobePProgram](#), [GetFilterPProgram](#), [ClearStrobePProgram](#), [ReadFilterPProgram](#).

2.63 ReadFilterPProgram

Function ReadFilterConsole reads the data associated with a part program event.

Syntax:

```
ret_code := ReadFilterPProgram (Process, PProgramData) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameters:

PProgramData (ANY) Structure to read data

Possible calls:

ReadFilterPProgram (INT,MFunc_FilterStruct)
ReadFilterPProgram (INT,SFunc_FilterStruct)
ReadFilterPProgram (INT,TFunc_FilterStruct)
ReadFilterPProgram (INT,CONMOV_FilterStruct)
ReadFilterPProgram (INT,ENDMOV_FilterStruct)
ReadFilterPProgram (INT,PSEUDO_FilterStruct)
ReadFilterPProgram (INT,RQP_FilterStruct)
ReadFilterPProgram (INT,RQT_FilterStruct)
ReadFilterPProgram (INT,TOU_FilterStruct)
ReadFilterPProgram (INT,PROBE_FilterStruct)
ReadFilterPProgram (INT,GTS_FilterStruct)
ReadFilterPProgram (INT,GTA_FilterStruct)
ReadFilterPProgram (INT,SPG_FilterStruct)
ReadFilterPProgram (INT,TCP_FilterStruct)
ReadFilterPProgram (INT,DUAL_FilterStruct)

Execution mode

Immediate

Use:

When the system has notified a part program event (whose notification has been requested with the SetFilterPProgram function), it is possible to read the data associated with it.

For each part program event there is an structure suitable for the required data, and hence the structure implicitly defines the event associated with it; for this reason, the read function only has to supply the *PProgramData* structure, and it need not supply the event.



To be informed of a part program event and be able to read the data, it is necessary to enable the required management beforehand by means of the SetFilterPProgram function. Reading the data without enabling the part program filter management function may yield non-significant values.

Through the GetStrobePProgram function it is possible to determine the part program event notified and present in the system.

If the function GetStrobePProgram returns the presence of event **f_M_FUNCTION** the required data must be read in a type **MFunc_FilterStruct** structure organised as follows:

Field	Type	Description
TypeOfMcode	INT	M function type
MCode	INT	M function number
ExpMode	INT	M function number if it is expedite
MParam	Array of LREAL	Parameters associated to M functions

TypeOfMcode	Mnemonic	Description
0	(PRELUDE)	M function previous to movement
1	(EXPEDITE)	M function contemporary to movement
2	(POSTLUDE)	M function subsequent to movement
128	(PREFETCH)	Execution in Prefetch stage (to be added to previous values)

If the function GetStrobePProgram returns the presence of event **f_S_FUNCTION** the required data must be read in a type **SFunc_FilterStruct** structure organised as follows:

Field	Type	Description
SpindleMode	INT	Execution mode G96/G97
SValue	LREAL	Programmed value with the S. In G97 it is expressed in rev/min, in G96 units/min
SSL	LREAL	SSL value (upper limit) associated
CnfgValue	LREAL	Programmed value with the S in machine units configuration

CnfgValue	Mnemonic	Description
0	G97_MODE	S programmed in G97
1	G96_MODE	S programmed in G96

If function GetStrobePProgram returns the presence of event **f_T_FUNCTION** the relative data must be read in a type **TFunc_FilterStruct** structure organised as follows:

Field	Type	Description
ToolProgMode	INT	T programming mode
OffsetNumber	INT	Offset number associated
NumSlave	INT	Tool slave number
Tmt	LREAL	TMT value associated
ToolCode	STRING	Tool code (string)
AxisId	Array of INT	Id axes where apply offsets
AxisDir	Array of LREAL	Offsets application direction +/- 1.0
Slaveld	Array of INT	Tool slave identifiers

ToolProgMode	Mnemonic	Description
1	T_tool_simple	T programmed without offset
2	T_tool_offset	T programmed with offset
3	T_tool_0	T programmed with offset 0
4	T_0	T0.0 programmed
5	T_0_0	T0.0 programmed
6	T_0_offset	T0 programmed with offset
10	MULTITOOL	Multi tool

If function GetStrobePProgram returns the presence of event **f_CONSENT_TO_MOVE** the relative data must be read in a type **CONMOV_FilterStruct** structure organised as follows:

Field	Type	Description
TypeOfMove	WORD	Movement type
FixedCycle	WORD	Fixed cycles in the movement
Modality	WORD	Movement mode
Feed	LREAL	Programmed feed
AxisId	Array of INT	Axes identifier to be moved
AxisPos	Array of LREAL	Final position for every axis

TypeOfMove	Mnemonic	Description
0x0001	G00_RAPID	Movement in rapid G0 linear
0x0002	G01_LINEAR	Movement in working G1 linear
0x0004	G02_03_CIRCLE	Movement in working G2 or G3 circular
0x0008	G12_13_14_CIRC3D	Movement in G61 splines
0x2000	G61_HSM	Movement in continuous G27
0x4000	G27_DYNAMIC	Movement in continuous G28
0x8000	G28_DYNAMIC	Movement in point-point G29

FixedCycle	Mnemonic	Description
0x0001	G81_DRILLING	Drilling cycle
0x0002	G82_SPOTFACING_DRILL	Spot-facing cycle
0x0004	G83_DEEPDRILL	Deep drill cycle
0x0008	G84_TAPPING	Tapping cycle
0x0010	G85_REAMING	Reaming cycle
0x0020	G86_BORING	Boring cycle
0x0040	G89_SPOTFACING_BOKE	Boring cycle with spot facing
0x2000	G74_POINTPROBEFIX	Point measuring cycle with displacement
0x4000	G73_CIRCLEPROBE	Circle probing cycle
0x8000	G72_POINTPROBE	Point probing cycle

Modality	Mnemonic	Description
0x0001	CONTINUOUS_JOG	Movement in manual continuous
0x0002	INCREMENTAL_JOG	Movement in manual continuous
0x0004	HOMING	Zero setting movement
0x0020	JOG_RETURN	Movement of return on the profile
0x4000	MAS_BLOCK	Movement in MAS status
0x8000	PROGRAM_BLOCK	PartProgram movement

If function GetStrobePProgram returns the presence of event **f_END_OF_MOVE** the relative data must be read in a type **ENDMOV_FilterStruct** structure organised as follows:

Field	Type	Description
ToolLifeValue	LREAL	Current value of tool life

If function GetStrobePProgram returns the presence of event **f_PSEUDO_AX_FUNCTION** the relative data must be read in a type **PSEUDO_FilterStruct** structure organised as follows:

Field	Type	Description
AxisId	Array of INT	Pseudo axes identifier
AxisPos	Array of LREAL	Programmed value for every axis

If function GetStrobePProgram returns the presence of event **f_RQP_FUNCTION** the relative data must be read in a type **RQP_FilterStruct** structure organised as follows:

Field	Type	Description
OffsetNumber	INT	OffsetNumber
ToolCode	STRING	Tool code (string)
ToolLenUpd	Array of INT	Length update request flag
ToolLen	Array of LREAL	Lengths in configuration unit
ProgLen	Array of LREAL	Requested lengths value (programmed)
ToolDiamUpd	Array of INT	Diameter update request flag
ToolDiam	Array of LREAL	Diameters in configuration units
ProgDiam	Array of LREAL	Requested diameter value (programmed)

If function GetStrobePProgram returns the presence of event **f_RQT_FUNCTION** the relative data must be read in a type **RQT_FilterStruct** structure organised as follows:

Field	Type	Description
OffsetNumber	INT	OffsetNumber
ToolCode	STRING	Tool code (string)
ToolLenUpd	Array of INT	Length update request flag
ToolIncLen	Array of LREAL	Lengths in configuration unit
ProgIncLen	Array of LREAL	Requested lengths value (programmed)
ToolDiamUpd	Array of INT	Diameter update request flag
ToolIncDiam	Array of LREAL	Diameters in configuration units
ProgIncDiam	Array of LREAL	Requested diameter value (programmed)

If function GetStrobePProgram returns the presence of event **f_TOU_FUNCTION** the relative data must be read in a type **TOU_FilterStruct** structure organised as follows:

Field	Type	Description
ToolCode	STRING	Tool code (string)

If function GetStrobePProgram returns the presence of event **f_PROBE_FUNCTION** the relative data must be read in a type **PROBE_FilterStruct** structure organised as follows:

Field	Type	Description
ProbeType	WORD	Probing type
AxisId	Array of INT	Axis identifier in probing
AxisPos	Array of LREAL	Position probe
AxisPos	Array of LREAL	Deviation from theoretical position (G74)

ProbeType	Mnemonic	Description
0x0001	G72_PROBING	Probing in G72 (point)
0x0002	G73_PROBING	Probing in G73 (circumference)
0x0004	G74_PROBING	Probing in G74 (displacement point)

If function GetStrobePProgram returns the presence of event **f_SHARE_FUNCTION** relative data must be read in a type **SHARE_FilterStruct** structure organised as follows:

Field	Type	Description
SharingMode	INT	Sharing mode
SharingType	INT	Type of shared object (axis, Spindle)
AxisId	Array of INT	Shared spindle/axis identifier

SharingMode	Mnemonic	Description
0	SHARE_TERMINATE	Ends sharing
1	SHARE_EXECUTE	Activate sharing
2	SHARE_RELEASE	Sharing axis release
3	SHARE_GET	Gets shared axis
4	SHARE_GET_EXCLUSIVE	Exclusively gets shared axis

SharingType	Mnemonic	Description
0	SHARE_AXIS	Sharing axis
1	SHARE_SPINDLE	Sharing spindle

If function GetStrobePProgram returns the presence of event **f_SPG_FUNCTION** the relative data must be read in a type **SPG_FilterStruct** structure organised as follows:

Field	Type	Description
Environment	WORD	Environment requesting enabling/disabling
SPGPhase	INT	Execution phase SPG
ErrCode	DINT	Execution error SPG
ProgName	STRING	Programme name to be selected

SPGPhase	Mnemonic	Description
0	BEFORE_SPG	Before starting programme activation
1	AFTER_SPG_OK	After activation without errors
2	AFTER_SPG_OK	After activation with errors
3	BEFORE_RELEASEPROG	Before starting programme deactivation

If function GetStrobePProgram returns the presence of event **f_GTA_FUNCTION** the relative data must be read in a type **GTA_FilterStruct** structure organised as follows:

Field	Type	Description
GTAType	WORD	Activation mode of GTA
AxisId	Array of INT	Axes identifier to be managed
AxisOffset	Array of INT	Axes identifiers for offset application

GTAType	Mnemonic	Description
-1	RELEASE_AXIS	Axis release
0	GET_AXIS_T0	Axis acquisition with origins lost
1	GET_AXIS_T1	Axis acquisition with origins maintained (1)
2	GET_AXIS_T2	Axis acquisition with origins maintained (2)

If function GetStrobePProgram returns the presence of event **f_TCP_FUNCTION** relative data must be read in a type **TCP_FilterStruct** structure organised as follows:

Field	Type	Description
TcpMode	INT	TCP enabling/disabling mode
TcpType	INT	TCP type
KinId	INT	Active kinematics number
KinType	INT	Active kinematics type
ToolHol	INT	Toolholder number enabled
NumAx	INT	Number of axes involved in the TCP
AxisId	Array of INT	Axes to be managed identifier

TcpMode	Mnemonic	Description
1	TCP_ACTIVATE	TCP enabling
2	TCP_DEACTIVATE	TCP disabling
3	TCP_CHANGE	TCP change in continuous (tcp,...)

TcpType	Mnemonic	Description
1	TCP_NORMAL	TCP without additional virtual axes
5	TCP_TOOLDIR	TCP with directional axis tool

KinType	Mnemonic	Description
1	TCP_DTWIST	DoubleTwist or MAKA kinematics
2	TCP_VERSOR	Versors only kinematics
10	TCP_HSM	Kinematics in HSM mode

If function GetStrobePProgram returns the presence of event f_DUAL_FUNCTION the relative data must be read in a type DUAL_FilterStruct structure organised as follows:

Field	Type	Description
DualType	INT	Dual axis type applied
Command	INT	Enabling/disabling mode
MastId	INT	Master axis id
NumAx	INT	Slave axes number
Slaveld	Array of INT	Slave axes id
FollRate	Array of LREAL	Following rate

DualType	Mnemonic	Description
1	DUAL_UDA	Dual in UDA mode
2	DUAL_SDA	Dual in SDA mode
3	DUAL_XDA	Dual in DA mode

Command	Mnemonic	Description
1	DUAL_DEFINE	Dual enabling command
2	DUAL_UNDEF	Dual disabling command
3	DUAL_ACT	Following enabled (XDA)
4	DUAL_DISACT	Following disabled (XDA)
5	DUAL_CHG	Following ratio change (XDA)

If function GetStrobePProgram returns the presence of event f_EOB_FUNCTION the relative data must not be read because no data is associated to the event.

Return values:

The following table lists all the possible values of the *ret_code* variable:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150020	Process parameter wrong (outside range)
0x00150021	Process not existent

Example:

```

Ret      : dword;
FdConmov : CONMOV_FilterStruct ;
FdEndmov : ENDMOV_FilterStruct;
Event    : dint ;

Ret := GetStrobePProgram(1, Event);
CASE Event OF
  f_CONSENT_TO_MOVE :
    Ret := ReadFilterPProgram (1, FdConmov );
  f_END_OF_MOVE :
    Ret := ReadFilterPProgram (1, FdEndmov );
END_CASE;

```

See also:

[SetFilterPProgram](#), [AckStrobePProgram](#), [ClearStrobePProgram](#), [GetStrobePProgram](#).

2.64 AckStrobePProgram

The AckStrobePProgram function informs the system that a PartProgram event has been acquired by the PLC and communicates the result thereof.

Syntax:

```
ret_code := AckStrobePProgram (Process, Event, AckNack) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>Event</i> (DINT)	Type of event whose outcome is notified
<i>AckNack</i> (BOOL)	Acknowledge given/not given to system for event notified

Execution mode:

Immediate

Use:

If the management of PartProgram filters has been enabled (see SetFilterPProgram), any part program request generated in the system (see Event) is communicated to the PLC. The moment it has acquired the communication (see GetStrobePProgram), the PLC must inform the system of the acquisition by giving out an acknowledge through the variable *AckNack*.

Upon receiving this acknowledge, the system is able to notify further subsequent PartProgram requests.



The system will freeze if the PLC does not reply to a PartProgram event with the AckStrobePProgram function. Needless to say, this is so provided that the PLC has activated the filter management function on PartProgram events.

The *Event* parameter defines the event mode and will be:

Event	Value	Description
f_M_FUNCTION	0x00000001	Acknowledge for function M
f_S_FUNCTION	0x00000002	Acknowledge for function S
f_T_FUNCTION	0x00000004	Acknowledge for function T
f_CONSENT_TO_MOVE	0x00000008	Acknowledge for movement consent
f_END_OF_MOVE	0x00000010	Acknowledge for end movement
f_PSEUDO_AX_FUNCTION	0x00000020	Acknowledge for pseudo axes
f_EOB_FUNCTION	0x00000040	Acknowledge for End Of Block
f_RQP_FUNCTION	0x00000080	Acknowledge for function RQP
f_RQT_FUNCTION	0x00000100	Acknowledge for function RQT
f_TOU_FUNCTION	0x00000200	Acknowledge for function TOU
f_PROBE_FUNCTION	0x00000400	Acknowledge for probing cycle result
f_GTS_FUNCTION	0x00000800	Acknowledge for function GTS
f_SPG_FUNCTION	0x00001000	Acknowledge for function SPG
f_GTA_FUNCTION	0x00002000	Acknowledge for function GTA
f_TCP_FUNCTION	0x00004000	Acknowledge for function TCP
f_DUAL_FUNCTION	0x00008000	Acknowledge for dual axes function

The *AckNack* parameter defines the reply mode and will be:

AckNack	Value	Description
ACK	true	Request accepted by the PLC
NACK	false	Request not accepted by the PLC

If the PLC does not acknowledge the request, the system will generate an error message.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Event parameter wrong
0x00150020	Process parameter wrong (outside range)
0x00150021	Process not existent

Example:

```
Ret : dword;
Ret := AckStrobePProgram (1, f_M_FUNCTION, ACK) ;
```

See also:

[SetFilterPProgram](#), [GetStrobePProgram](#), [ClearStrobePProgram](#), [ReadFilterPProgram](#)

2.65 ClearStrobePProgram

Function ClearStrobePProgram resets (for the PLC only) the presence of a PartProgram event.

Syntax:

```
ret_code := ClearStrobePProgram (Process, Event) ;
```

Input parameters:

Process (INT) Process number [1..24]

Event (DINT) PartProgram event to be deleted

Execution mode:

Immediate

Use:

This function deletes the presence of a PartProgram event notified to the PLC. The moment a PartProgram event is generated, the system keeps it present and available for the PLC until the PLC acknowledges its acquisition through the AckStrobePProgram function. At this point, the system is ready to generate further console requests.

If an event no longer to be notifiable (i.e., the GetStrobePProgram function will not notify any console event present to the PLC) and at the same time maintain the PartProgram event management frozen, use this function.

The *Event* parameter identifies the PartProgram event to be reset. Its values may be:

Event	Value	Description
f_M_FUNCTION	0x00000001	For function M event
f_S_FUNCTION	0x00000002	For function S event
f_T_FUNCTION	0x00000004	For function T event
f_CONSENT_TO_MOVE	0x00000008	For movement consent event
f_END_OF_MOVE	0x00000010	For movement end event
f_PSEUDO_AX_FUNCTION	0x00000020	For pseudo axes event
f_EOB_FUNCTION	0x00000040	For End Of Block event
f_RQP_FUNCTION	0x00000080	For function RQP event
f_RQT_FUNCTION	0x00000100	For function RQT event
f_TOU_FUNCTION	0x00000200	For function TOU event
f_PROBE_FUNCTION	0x00000400	For probing cycle result event
f_SHARE_FUNCTION	0x00000800	For function SHARE event
f_SPG_FUNCTION	0x00001000	For function SPG event
f_GTA_FUNCTION	0x00002000	For function GTA event
f_TCP_FUNCTION	0x00004000	For function TCP event
f_DUAL_FUNCTION	0x00008000	For dual axes function event



The system will freeze if the PLC does not reply to a PartProgram event with the AckStrobePProgram function. Assuming that the PLC has activated the filter management function on PartProgram events.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Event parameter wrong
0x00150020	Process parameter wrong (outside range)
0x00150021	Process non-existent

Example:

```
Ret : dword;  
  
Ret := ClearStrobePProgram (1, f_M_FUNCTION) ;
```

See also:

[SetFilterPProgram](#), [GetStrobePProgram](#), [AckStrobePProgram](#), [ReadFilterPProgram](#)

2.66 GetJogIncrement

Function GetJogIncrement reads Jog Increment value.

Syntax:

```
ret_code := GetJogIncrement (Process, JogValue) ;
```

Input parameters:

Process (INT) Process number [1..24]

Output parameter:

JogValue (LREAL) Jog Increment value

Execution mode:

Immediate

Use:

This function returns the current of the incremental JOG (manual axis movements).

The value is written in the JogValue variable.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example:

```
Ret : DWORD ;
Jog : LREAL ;
Ret := GetJogIncrement (1, Jog) ;
```

See also:

[SetJogIncrement](#), [ReadProcessInfo](#)

2.67 SetJogIncrement

Function SetJogIncrement sets Jog Increment.

Syntax:

```
ret_code := SetJogIncrement (Process, JogValue) ;
```

Input parameters:

Process (INT)	Process number [1..24]
JogValue (LREAL)	Jog Increment value

Execution mode:

Immediate

Use:

This function forces the incremental manual jog value on the declared process.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example

```
Ret : DWORD ;  
Ret := SetJogIncrement (1, 0.01) ;
```

See also

[GetJogIncrement](#), [ReadProcessInfo](#)

2.68 SetManualOverride

Function SetManualOverride selects manual feed rate override percentage.

Syntax:

```
ret_code := SetManualOverride (ExecMode, ExecStatus, Process, OverridePercent,  
                          OverrideDirection) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>OverridePercent</i> (INT)	Override percentage
<i>OverrideDirection</i> (INT)	Direction manual motion
<i>OverrideMode</i> (INT)	Override application mode

Execution mode:

Wait / NoWait

Use:

Sets the feed rate override percentage used by manual motion. The parameter *OverridePercent* can have values between 0 and 10000 (0% and 100%), the past value is defined equal to 1/100 of %.

The manual movement direction depends on the *OverrideDirection* parameter, which may have positive values, POSITIVE_DIR (0) for manual movement in positive direction and NEGATIVE_DIR (1) for manual movement in negative direction.

Override application mode (defined by the *OverrideMode* variable) determines how to apply percentage values and direction. It may have the following values:

- ▷ MFO_INCREMENTAL (0) override value increases/decreases of a value equal to the *OverrideDirection*. If the parameter *OverrideDirection* is POSITIVE_DIR, the override increases, decreasing if *OverrideDirection* is NEGATIVE_DIR.
- ▷ MFO_ABSOLUTE (1) activates the override with a percentage defined by *OverridePercent* and a movement direction defined by *OverrideDirection*.
- ▷ MFO_DIRECTION (2) activates only the movement direction defined by the *OverrideDirection* parameter. As *OverridePercent* is not considered, the parameter maintains the override percentage active.
- ▷ MFO_PERCENT (3) activates only the override percentage define by *OverridePercent*. As *OverrideDirection* is not considered, the parameter keeps the movement direction active at that moment.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example:

```
Ret      : dword;
Dummy   : dword;

Ret := SetManualOverride (MODE_WAIT,     Dummy,     1,      7000,      NEGATIVE_DIR,
MFO_ABSOLUTE) ;
```

See also:

[ReadProcessInfo](#), [SetSpindleOverride](#), [SetFeedOverride](#), [SetRapidOverride](#)

2.69 SetMultiBlockRetrace

Function SetMultiBlockRetrace sets the program retrace functionality (MBR).

Syntax:

```
ret_code := SetMultiBlockRetrace (ExecMode, ExecStatus, Process, Mode) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Mode</i> (WORD)	MBR execution mode

Execution mode:

Wait / NoWait

Use:

If retrace is active, this function defines whether to send or not the part program auxiliary functions to the machine logic.

Mode	Mnemonic	Description
0x0001	MBR_AUX_RUN	Auxiliary functions passed to the PLC in retrace.
0x4000	MBR_DEACTIVATE	Deactivates retrace function
0x8000	MBR_ACTIVATE	Activates retrace function

The MBR_ACTIVATE and MBR_DEACTIVATE instructions must be sent one by one.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number
0x00150025	Mode parameter wrong

Example:

```
Ret      : dword;
Dummy   : dword;
Ret := SetMultiBlockRetrace (MODE_WAIT, Dummy, 1, 0) ;
```

See also:

[ReadProcessInfo](#)

2.70 SetSearchInMemory

The function SetSearchInMemory defines the mode for the search in memory (RCM)

Syntax:

```
ret_code := SetSearchInMemory (ExecMode, ExecStatus, Process, Mode) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

Mode (WORD) RCM execution mode

Execution mode

Wait / NoWait

Use:

If the search in memory is active, this function defines how the auxiliary functions have to be managed; that is if they have to be memorized and sent to the logic only once at the end of the search or whenever they are found in the part program.

The modality to be used can be selected through the *Mode* parameter that can have the following values:

ProcSts	Mnemonic	Description
0x0001	RCM_M_RUN	M function passed to PLC during the search
0x0002	RCM_T_RUN	T function passed to PLC during the search
0x0004	RCM_S_RUN	S function passed to PLC during the search
0x0008	RCM_AX_RUN	axes function passed to PLC during the search

If parameter *Mode* = 0, the auxiliary functions are memorized and sent to the logic only at the end of the search. The SRC_ACTIVATE and SRC_DEACTIVATE requests should be given singularly, not together with the others.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example:

```
Ret      : dword;
Dummy   : dword;
Ret := SetSearchInMemory (MODE_WAIT, Dummy, 1, 0) ;
```

See also:

[ReadProcessInfo](#)

2.71 SetRapidOverride

The function SetRapidOverride selects rapid Feed Override percentage.

Syntax:

```
ret_code := SetRapidOverride (ExecMode, ExecStatus, Process, OverridePercent) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

OverridePercent (INT) Override percentage

Execution mode:

Wait / NoWait

Use:

Forces the rapid feed rate with a value equal to the percentage of the configured maximal rapid feed rate. This function only effects the rapid feed rate when the override rapid is enabled (Boolean process variable URL=1). This variable can be set by the operator in the part program set up menu or by PLC through VAR_Write).

The parameter *OverridePercent* can have values between 0 and 10000 (0% e 100%), the past value is defined equal to 1/100 of %.

The rapid feed rate cannot be increased to a value higher than the configured rapid feed rate (100 %).

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example:

```
Ret      : dword;
Dummy   : dword;
Ret := SetRapidOverride (MODE_WAIT, Dummy, 1, 3000) ;
```

See also:

[ReadProcessInfo](#), [SetSpindleOverride](#), [SetManualOverride](#), [SetFeedOverride](#)

2.72 SetFeedOverride

The function SetFeedOverride selects Feed Override percentage.

Syntax:

```
ret_code := SetFeedOverride (ExecMode, ExecStatust, Process, OverridePercent) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

OverridePercent (INT) Override percentage

Execution mode:

Wait / NoWait

Use:

This function used to control the feed rate of the axes when not in G00. The feed rate is limited by the system to the configured rapid feed rate for every axis. The true feed rate percentage must be multiplied with 100 to obtain the controls internal resolution, the previous value is defined with granularity = 1/100 of %.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example:

```
Ret      : dword;
```

```
Dummy   : dword;
```

```
Ret := SetFeedOverride (MODE_WAIT, Dummy, 1, 10000) ;
```

See also:

[ReadProcessInfo](#), [SetSpindleOverride](#), [SetManualOverride](#), [SetRapidOverride](#)

2.73 SetSpindleOverride

The function SetSpindleOverride selects Spindle Override percentage.

Syntax:

```
ret_code := SetSpindleOverride (ExecMode, ExecStatus, Process, OverridePercent) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Process (INT) Process number [1..24]

OverridePercent (INT) Override percentage

Execution mode:

Wait / NoWait

Use:

This function is used to force the spindle speed override value in the systems process display. PLC uses the *OverridePercent* value to control the spindle RPM in G97 or the tool cutting speed in G96. The past value is defined equal to 1/100 of %.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example:

```
Ret      : dword;  
Dummy   : dword;
```

```
Ret := SetSpindleOverride (MODE_WAIT, Dummy, 1, 7500) ;
```

See also:

[ReadProcessInfo](#), [SetRapidOverride](#), [SetManualOverride](#), [SetFeedOverride](#)

2.74 SpindleGet

Function SpindleGet calls the use of a spindle shared with the PLC.

Syntax:

```
ret_code := SpindleGet (ExecMode, ExecStatus, Process, Mode, SpindleId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number[1..24]
<i>Mode</i> (INT)	Use mode
<i>SpindleId</i> (INT)	Identifier and spindle [1..64]

Execution mode:

Wait / NoWait

Use:

This function allows the use of a spindle (*SpindleId*) within a process. Spindle use mode must be defined using the *Mode* parameter having the following values:

Mode	Mnemonic	Description
3	SHARE_GET	Shared use
4	SHARE_GET_EXCLUSIVE	Exclusive use

The exclusive use is compulsory in tapping/boring operations during which there are the inversion/stop of the spindle. This instruction can be called only if the process is in IDLE, MBR and MAS status.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example

```
Ret : dword;
(* Requires ID 2 spindle use on process 1 *)
Ret := SpindleGet (MODE_NOWAIT, UL0, 1, 2);
```

See also:

SpindleRelease

2.75 SpindleRelease

Function SpindleRelease releases a spindle shared with the PLC.

Syntax:

```
ret_code := SpindleRelease (ExecMode, ExecStatus, Process) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]

Execution mode:

Wait / NoWait

Use:

This function releases a spindle and can be called only if the process is in IDLE, MBR and MAS status.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example:

```
Ret : dword;  
(* Releases the spindle from process *)  
Ret := SpindleRelease (MODE_NOWAIT, UL0, 1);
```

See also:

[SpindleGet](#)

2.76 TeachPendOFF

Function TeachPendOFF disables the serial Teach Pendant for a process.

Syntax:

```
ret_code := TeachPendOFF (Process) ;
```

Input parameters:

Process (INT) Process number [1..24]

Execution mode:

Immediate

Use:

This function disables the Teach Pendant on a serial line (this device has to be configured in ODM) within the process specified from the *Process* variable. The process has to be previously enabled using the TeachPendON function. All processes related to the Teach Pendant are disabled if the process number value is 0.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number
0x00170018	Wrong command parameter

Example:

```
Ret : dword;  
Ret := TeachPendOFF (1) ;
```

See also:

TeachPendON

2.77 TeachPendON

The TeachPendON function enables the use of the serial Teach Pendant within a process.

Syntax:

```
ret_code := TeachPendON (Process, Mode) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>Mode</i> (WORD)	Activation mode

Execution mode:

Immediate

Use:

This function disables the Teach Pendant on a serial line (this device has to be configured in ODM) within the process specified from the *Process* variable. *Mode* defines the Teach Pendant effectiveness according to the following table:

Mode	Mnemonic	Description
0x00000001	TEACH_AUTOMode	Enables automatic Teach Pendant mode
0x00000002	TEACH_MANMode	Enables manual Teach Pendant mode
0x00000004	TEACH_HWNDMode	Enables hand wheel Teach Pendant mode

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number
0x00170018	Wrong command parameter

Example:

```
Ret : dword;
Ret := TeachPendON (1, TEACH_AUTOMode) ;
```

See also:

TeachPendOFF

2.78 TOOL_ActivOffset

The function TOOL_ActivOffset requests activation of a tool offset.

Syntax:

```
ret_code := TOOL_ActivOffset (ExecMode, ExecStatus, Process, ToolCode, ToolOfsNum, Offset) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>ToolCode</i> (STRING)	Tool identifier
<i>ToolOfsNum</i> (INT)	Offset number to be applied
<i>Offset</i> (TOOLOffs_Struct)	Offset to be applied

Execution mode

Wait / NoWait

Use:

The M06 code (or any other M code) can be used to activate the offsets related to the new tool. The selected M code for tool offset activation is given the tool activation attribute in ODM. When the NC reads the M06 code, it goes into a halt state (stop of block pre-calculation) in which new offsets can be applied.

Follow this step by step procedure to change a tool and activate a new offset:

1. Decide in the logic, which offset(s) must be used for the selected tool.
2. Read the offset values from the offset table and write them to the axis table in the appropriate axis fields ToolOffs, OrigOfs and G92Ofs of the structure *Offset* of type TOOLOffs_Struct. Define tool orientation in *Orient* field (0..8).
3. Calculate the total offset fields (TotalOfs) of the *Offset* structure (add tool offset, offset G92 and value of origin).

In field CtrlWord of the *Offset* structure, enter the offset activation mode. This Control Word may contain the following values:

Mnemonic	Value (HEX)	Description
USE_TOTOFFS	0x0001	Activates TotalOfs input to structure
USE_TOOLLEN	0x0002	Activates tool length
USE_ORIGIN	0x0004	Activates origin
USE_G92	0x0008	Activates offset G92

If bit USE_TOTOFFS is set, the total offset present in field TotalOfs of the structure will be activated on the axis, and this will be done irrespective of the values of the other components of total offset; otherwise, total offset will be given as the sum of Origin (OrigOfs), Offset G92 (G92Ofs) and tool length (ToolOfs).

After the function ACTOFFS returns a 0 value, call the function PPRESUME to resume the execution of the part program with the new offsets active.

For standard applications the steps 2 and 3 can be executed in one single function call TOOL_CalcOffset.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x00170008	Axis not belonging to process
0x001700A8	Orientation wrong
0x0016002B	Axis not configured
0x0016002C	Axis with Id not admitted
0x0019002C	Offset wrong

Example

```

Ret      : dword;
Dummy    : dword;
ToolOffset : TOOLOffs_Struct;
ToolId   : STRING := 'TOOL 1' ;

ToolOffset.CtrlWord[0] = USE_TOTOFFS ;
ToolOffset.Orient     = 1 ;

Ret := TOOL_ActiveOffset (MODE_WAIT, Dummy, 1, ToolId, 100, ToolOffset) ;

```

See also

AXIS_ActiveOffset, TOOL_CalcOffset, AXIS_CalcOffset.

2.79 TOOL_CalcOffset

The function TOOL_CalcOffset calculates, on the basis of offset, the offsets to be applied on the axes.

Syntax:

```
ret_code := TOOL_CalcOffset (Process, ToolOfsNum, AxForOffset, Offset) ;
```

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>ToolOfsNum</i> (INT)	Offset number to be applied
<i>AxForOffset</i> (TOOLcalcOffs_Struct)	Offset to be applied on axes

Output parameters

<i>Offset</i> (TOOLoffs_Struct)	Correction values to be applied
---------------------------------	---------------------------------

Execution mode

Wait

Use:

It is the task of the WinPLUS programmer to apply the correct tool offsets. The tool offset number will be communicated to WinPLUS in the part program interface task with the command of function T or can be found in the tool table in the field OffsNum. The way a tool offset is handled depends on the axes defined inside the structure AxForOffset of type TOOLcalcOffs_Struct.

AxisId = 0	there will be no offset calculated for this axis
Other values	an offset will be calculated for this axis

The activation/de-activation of the new offsets is done with a dedicated M code (in most cases M06). In ODM, this code is given the attribute of "offset activation". The WinPLUS program must now use the function TOOL_ActivOffset to activate the elaborated offsets.

The same M code is normally used to move the physical tool form the magazine to the spindle.

Mill/machining centre

For a standard mill application, normally just one length offset is applied in the axial tool direction, normally the spindle axis ("Z"). The above routine needs the physical axis identifier for the spindle axis and the offset number as inputs. The other axes are set to a zero value. The function reads all the required data from the tool offset table, calculates the effective offset for the axis and writes it to the axes table and returns the total offset values in a structure of type AXISOffs_Struct, where returns the total offset value calculated by the process (TotalOfs), the active origin value (OrigOfs) and G92 value for the relative axis (G92Ofs).

Lathe/cylindrical grinder

For a standard lathe or cylindrical grinder application, normally two length offsets are applied, one in the X direction and one in the Z direction. The above routine now needs the physical axis identifiers for the first and the second axis and the offset number as inputs. It reads all the required data from the tool offset table, calculates the offset for the axes writes them to the axes table and returns the total offset values in a structure of type `AXISOfs_Struct`, where the total offset value calculated by the process (`TotalOfs`), the tool offset value (`ToolOfs`), the active origin value (`OrigOfs`) and `G92` value (`G92Ofs`) is returned.

Generic applications

This routine requires as input the physical identifier of n (up to 5) axes on which to apply the offsets, and the offset number. This function reads all the data requested from the tool offset table, determines the actual offset for the axes, and returns the values in a type `AXISOfs_Struct` structure, where, axis by axis, it will supply the total offset value (`TotalOfs`) calculated by the process, the value of the tool offset present (`ToolOfs`), the value of the active origin (`OrigOfs`), and the `G92` value activated (`G92Ofs`).

This function executes following steps:

- › For every axis specified in structure `AxForOffset` and the axis specified by `ToolOfsNum`, this function reads the initial length (`InitialLen`), adds the length correction value (`ActChangeLen`), multiplies the sum by the negative value of `AxisDir` contained in structure `AxForOffset`, and writes the result as correction value for the tool length (`ToolOfs`) within the `Offset` output structure.
- › It will calculate the value of total offset (`TotalOfs`) and will also specify in the output structure the value of the active origin (`OrigOfs`) and the `G92` value activated (`G92Ofs`).
- › Always on the basis of `ToolOfsNum`, it calculates the tool diameter values to be applied and enters them in the field (`ToolDiam`) of the output structure. This value is given by the sum of the initial diameter value (`InitialDiam`) and the diameter correction value (`ActChangeDiam`).

The `Offset` structure can be used as an input parameter for `TOOL_ActiveOffset` system call which will physically apply the new offsets to the axis.

Disabling offsets

To de-activate a tool offset, use `TOOL_CalcOffset` in the same way as described above, entering a 0 value as Id axes input. The function will automatically write 0 values to the required output structure.

Return values

<code>ret_code (HEX)</code>	<code>Description</code>
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x00170008	Axis not belonging to process
0x0016002B	Axis not configured
0x0016002C	Axis with Id not admitted
0x0019002C	Offset wrong

Example:

```
Ret      : dword;
ToolList  : TOOLcalcOffs_Struct;
ToolOffset : TOOLOffs_Struct;

ToolList.AxisId[0]  = 1 ;
ToolList.AxisDir[0] = -1.0 ;

Ret := TOOL_CalcOffset (1, 100, ToolList, ToolOffset) ;
```

See also

[AXIS_ActiveOffset](#), [TOOL_ActiveOffset](#), [AXIS_CalcOffset](#).

2.80 AXIS_ActivOffset

Function AXIS_ActivOffset requests the application of the offsets calculated for given axes.

Syntax:

```
ret_code := AXIS_ActivOffset (ExecMode, ExecStatus, Process, Offset) ;
```

Input parameters

<i>ExecMode</i> (INT)	Execution mode of command [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Offset</i> (AXISOfts_Struct)	Offset to be applied to position

Execution mode

Wait / NoWait

Use:

This function may be used to activate an offset on one or more axes. A total offset or individual offset values, as specified, axis by axis, in an activation structure can be applied. This is an AXISOfts_Struct type structure containing, for each axis, the total offset value (TotalOfs) to be applied, the value of tool offset (ToolOfs), the value of the origin (OrigOfs) and value G92 (G92Ofs). A specific Control Word (CtrlWord) makes it possible to specify the offsets to be activated and the activation modes.

This Control Word may contain the following values:

Mnemonic	Value (HEX)	Description
USE_TOTOFFS	0x0001	Activates TotalOfs transferred to structure.
USE_TOOLLEN	0x0002	Activates tool length
USE_ORIGIN	0x0004	Activates origin
USE_G92	0x0008	Activates offset G92

If bit USE_TOTOFFS has been set, the total offset present in the TotalOfs field of the structure will be applied to the axis, and this will be done irrespective of the value of the other components of TotalOfs; otherwise, total offset will be given by the sum of:

- Value of active origin (OrigOfs)
- Offset G92 active (G92Ofs)
- Active tool length offset (ToolOfs)

The *Offset* (AXISOfts_Struct) structure may be either calculated arbitrarily or loaded by the process through function AXIS_CalcOffset.

Return values

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent
0x00170008	Axis does not belong to process
0x0016002B	Axis not configured
0x0016002C	Axis ID not admissible

Example

```

Ret      : dword;
Dummy    : dword;
AxisOffset : AXISOffs_Struct;

AxisOffset.CtrlWord[0] = USE_TOTOFFS ;
AxisOffset.CtrlWord[1] = USE_TOTOFFS ;

Ret := AXIS_ActiveOffset (MODE_WAIT, Dummy, 1, AxisOffset) ;

```

See also

[AXIS_CalcOffset](#), [TOOL_ActiveOffset](#), [TOOL_CalcOffset](#).

2.81 AXIS_CalcOffset

The function AXIS_CalcOffset requests to calculate the offsets to be applied for axes.

Syntax:

```
ret_code := AXIS_CalcOffset (Process, AxForOffset, Offset) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>AxForOffset</i> (AXIScalcOffs_Struct)	Axes whose offset must be calculated

Output parameters

<i>Offset</i> (AXISOffs_Struct)	Offset to be applied to axes
---------------------------------	------------------------------

Execution mode

Wait

Use

This function can be used to calculate the total offset of an axis. The total offset is the sum of:

- active origin value (OrigOfs)
- active G92 offset (G92Ofs)
- active tool length offset (ToolOfs)

At the output, the function will return a type AXISOffs_Struct structure, in which, for the individual axes, it will supply the value of total offset (TotalOfs) calculated by the process, the value of the tool offset present (ToolOfs), the value of the active origin (OrigOfs), and the G92 value activated (G92Ofs). To activate the total new offset, use the function AXIS_ActiveOffset. Normally AXIS_ActiveOffset is used in conjunction with a dedicated M code ('activates offset' attribute). This M code (which WinPLUS has to acknowledge) stops the part program execution. The part program execution will be continued (with the new offsets) when WinPLUS calls the system function PPRESUME.

Return values

The following table gives the values that may be assumed by variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent
0x00170008	Axis not belonging to process
0x0016002B	Axis not configured
0x0016002C	Axis with Id not admitted

Example

```
Ret      : dword;
AxisList  : AXISCalcOffs_Struct;
AxisOffset : AXISOffs_Struct;

AxisList.AxisId[0] = 1 ;
AxisList.AxisId[1] = 2 ;

Ret := AXIS_CalcOffset (1, AxisList, AxisOffset) ;
```

See also

[AXIS_ActiveOffset](#), [TOOL_ActiveOffset](#), [TOOL_CalcOffset](#).

2.82 TOOL_GetSlaveID

The function TOOL_GetSlaveID sets the slave tools.

Syntax:

```
ret_code := TOOL_GetSlaveID (Process, StartIndex, NumberOfIDs, SlaveIDs) ;
```

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>StartIndex</i> (INT)	Index of first tool from which to start reading
<i>NumberOfIDs</i> (INT)	Number of slave tools to read

Output parameters

<i>SlaveIDs</i> (array of WORD)	Array containing slave tools identifiers
---------------------------------	--

Execution mode:

Wait

Use:

The Function in question must be used in order to acquire the identifiers of the slave tools concerned in programming of the multi-tool T function.

Slave tool management is activated the moment type T30/31, 32 part program line is activated, in which tool 30 is associated with its slave tools 31 and 32.

Slave tool management must be performed at part program filter level, following the tool activation command indicating the slave tools.

Return values

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example:

```
Ret      : dword;
SlaveId  : array [0..SLAVES_TL-1] of WORD;

Ret := TOOL_GetSlaveId (1, 0, SLAVES_TL, AxisOffset) ;
```

2.83 VAR_Read

Function VAR_Read reads a process variable.

Syntax:

```
ret_code := VAR_Read (Process, Index, NumCha, Name, Value) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>Index</i> (INT)	Variable index
<i>NumCha</i> (INT)	Number of characters to be read for string type variables
<i>Name</i> (STRING)	Variable name

Output parameters

<i>Value</i> (ANY_ELEMENTARY)	Variable value
-------------------------------	----------------

Possible overload:

VAR_Read(INT, INT, INT, STRING, BOOL)
 VAR_Read(INT, INT, INT, STRING, BYTE)
 VAR_Read(INT, INT, INT, STRING, WORD)
 VAR_Read(INT, INT, INT, STRING, INT)
 VAR_Read(INT, INT, INT, STRING, DINT)
 VAR_Read(INT, INT, INT, STRING, LREAL)
 VAR_Read(INT, INT, INT, STRING, STRING)

Execution mode:

Immediate

Use:

Reads a process variable whose name has been input with parameter *Name*. This parameter must contain only the name of the variable, not its “index”; for example, for variable E100, string ‘E’ must be input as name, and value 100 as *Index*. The same consideration applies to variables H, HF, HC, L, LS, SN and SC. For string or character type variables, i.e., for LS, SC, HC and HS, the maximum number of characters to be read using field *NumCha* must be defined.

Return values:

The following table lists all the possible values that *ret_code* variable can have:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent
0x00170006	Variable format not consistent with reading variable type
0x00170007	Writing exceeds the variable size
0x0017000B	Non-existent variable

Example:

```
Ret      : dword;
StringVal : STRING ;
DoubleVal : LREAL ;
Ret := Var_Read (1, 100, 10, 'LS', StringVal) ;
Ret := Var_Read (1, 66, 0, 'E', DoubleVal) ;
```

See also:

[VAR_Write](#)

2.84 VAR_ReadArray

The VAR_ReadArray function reads a process variable.

Syntax

```
ret_code := VAR_ReadArray (Process, Index1, Index2, Index3, NumVar, Name, Value) ;
```

Input parameters

Process (INT)	Process number [1..24]
Index1 (INT)	First index variable
Index2 (INT)	Second index variable
Index3(INT)	Third index variable
NumVar(INT)	Number of elements to be read
Name (STRING)	Variable name

Output parameters

Value (ANY_ELEMENTARY)	Valore della variabile (su array)
------------------------	-----------------------------------

Possible overloads

VAR_ReadEx(INT, INT, INT, INT, INT, STRING, BYTE)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, WORD)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, INT)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, DWORD)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, DINT)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, LREAL)

Execution mode

Immediate

Use

Writes a process variable whose name has been specified with parameter *Name*. This parameter must contain only the name of the variable, not its “index”; for example, for variable E100, string ‘E’ must be specified by name, and value 100 as *Index*. The additional parameters *Index2* and *Index3* must be used for the variables (for example asset variables or ToolCenterPoint configuration) that require more indexes. Using *NumVar*, it is possible to define how many variables (in case of array type variables) must be read; if the variable is not an array-type, it necessary to pass to value 1.

This function is available only from the OPENcontrol V3.1.1.

Return values

The following table lists all values *ret_code* can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent
0x00170006	Size variable is not congruent with variable type of reading
0x00170007	Write exceeding the variable size
0x0017000B	Non-existent variable

Example

```
Ret      : dword;
DoubleVal : array [1..100] of LREAL ;

(* Reads 20 E variables starting from E66 *)
Ret := Var_ReadArray (1, 66, 0, 0, 20, 'E', DoubleVal[1]) ;
```

See also

[VAR_WriteArray](#)

2.85 VAR_Write

Function VAR_Write writes a process variable.

Syntax:

```
ret_code := VAR_Write ( Process, Index, NumCha, Name, Value ) ;
```

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>Index</i> (INT)	Variable index
<i>NumCha</i> (INT)	Number of characters to be written for string type variables
<i>Name</i> (STRING)	Variable name
<i>Value</i> (ANY_ELEMENTARY)	Variable value

Possible overloads:

VAR_Write(INT, INT, INT, STRING, BOOL)
 VAR_Write(INT, INT, INT, STRING, BYTE)
 VAR_Write(INT, INT, INT, STRING, WORD)
 VAR_Write(INT, INT, INT, STRING, INT)
 VAR_Write(INT, INT, INT, STRING, DINT)
 VAR_Write(INT, INT, INT, STRING, LREAL)
 VAR_Write(INT, INT, INT, STRING, STRING)

Execution mode:

Immediate

Use:

Writes a process variable whose name has been specified with parameter *Name*. This parameter must contain only the name of the variable, not its “index”; for example, for variable E100, string ‘E’ must be specified by name, and value 100 as *Index*. The same consideration applies to variables H, HF, HC, L, LS, SN and SC. For string or character type variables, i.e., for LS, SC, HC and HS, the maximum number of characters to be written using field *NumCha* must be defined.

Return values:

The following table lists all the possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x00170006	Variable format not congruent with write variable type
0x00170007	Read exceeding max. variable size
0x0017000B	Variable not existent

Example

```
Ret      : dword;
StringVal : STRING := 'Variabile LS';
DoubleVal : LREAL := 123.765;

Ret := Var_Write (1, 100, LEN (StringVal), 'LS', StringVal) ;
Ret := Var_Write (1, 66, 0, 'E', DoubleVal) ;
```

See also:

[VAR_Read](#)

2.86 VAR_WriteProtect

VAR_WriteProtect function writes a process of a Write Protect type variable.

Syntax

```
ret_code := VAR_WriteProtect (ExecMode, ExecStatus, Process, VarType, Value) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process Number [1..24]
<i>VarType</i> (INT)	Variable type
<i>Value</i> (BOOL)	Variable value

Execution Mode

Immediate

Use

The function writes a process variable having a *VarType* type. Variables that can be accessed by this function are “Write Protect” type and cannot be written by part program. These variables enable/disable some operational modalities of the addressed process. These variables can be read by reading functions as VAR_Read/VAR_ReadEx.

Types of accessible variables:

VarType	Mnemonic	Description
1	Var_DPS	Enables/disables Part Program Scroll
2	Var_URL	Enables/disables limited rapids
3	Var_UVR	Enables/disables rapid feed
4	Var_RAP	Enables/disables automatic Jog Return
5	Var_ERR	Enables/disables Part Program errors management
6	Var_DSB	Enables/disables block delete
7	Var_USO	Enables/disables Optional Stop
8	Var_HMP	Enables/disables automatic Homing
9	Var_RCNEAR	Enables/disables the search for the nearest program block

Return values

The following table lists all possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x0017000B	Not existent variable

Example

```
Ret : dword;
Status : dword;
Ret := Var_WriteProtect (MODE_WAIT, Status, Var_DSB, true) ;
```

See also

[VAR_Read](#), [VAR_ReadEx](#)

2.87 VAR_WriteArray

The function VAR_WriteArray writes a process variable.

Syntax

```
ret_code := VAR_WriteArray ( Process, Index1 ,Index2, Index3, NumVar, Name, Value ) ;
```

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>Index1</i> (INT)	First variable index
<i>Index2</i> (INT)	Second variable index
<i>Index3</i> (INT)	Third variable index
<i>NumVar</i> (INT)	Number of variables to write
<i>Name</i> (STRING)	Variable name
<i>Value</i> (ANY_ELEMENTARY)	Variable value

Possible overloads

VAR_WriteEx (INT,INT,INT,INT,INT,STRING,BYTE)
 VAR_WriteEx (INT,INT,INT,INT,INT,STRING,WORD)
 VAR_WriteEx (INT,INT,INT,INT,INT,STRING,INT)
 VAR_WriteEx (INT,INT,INT,INT,INT,STRING,DWORD)
 VAR_WriteEx (INT,INT,INT,INT,INT,STRING,DINT)
 VAR_WriteEx (INT,INT,INT,INT,INT,STRING,LREAL)

Execution mode

Immediate

Use

Writes a process variable whose name has been specified with parameter *Name*. This parameter must contain only the name of the variable, not its “index”; for example, for variable E100, string ‘E’ must be specified by name, and value 100 as *Index*. The additional parameters *Index2* and *Index3* must be used for the variables (for example asset variables or ToolCenterPoint configuration) that require several indexes. Using *NumVar*, it is possible to define how many variables (in case of array type variables) must be read; if the variable is not an array-type, it necessary to use value 1.

This function will be available from the OPENcontrol V3.1.1.

Return values

The following table lists all the values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent
0x00170006	Variable format is incongruent with the variable type being read
0x00170007	Reading exceeds the variable dimension
0x0017000B	Non-existent variable

Example

```
Ret      : dword;
DoubleVal : array [1..100] of LREAL ;

(* Writes 20 E variables starting from variable E66 *)
Ret := Var_WriteArray (1, 66, 0, 0, 20, 'E', DoubleVal[1]) ;
```

See also

[VAR_ReadArray](#)

2.88 VAR_ReadEx

VAR_ReadEx function reads a process variable.

Syntax

```
ret_code := VAR_Read (Process, Index1, Index2, Index3, NumCha, Name, Value) ;
```

Input parameters

Process (INT)	Process number [1..24]
Index1 (INT)	First variable index
Index2 (INT)	Second variable index
Index3(INT)	Third variable index
NumCha(INT)	Number of characters to read for string type variables
Name (STRING)	Variable name

Output parameters

Value (ANY_ELEMENTARY)	Variable value
------------------------	----------------

Possible overloads

VAR_ReadEx(INT, INT, INT, INT, INT, STRING, BOOL)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, BYTE)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, WORD)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, INT)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, DWORD)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, DINT)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, LREAL)
VAR_ReadEx(INT, INT, INT, INT, INT, STRING, STRING)

Execution mode

Immediate

Use

This function reads a process variable having an input name in the *Name* parameter. This parameter only contains the variable name and its “index”; for instance, in the E100 variable, ‘E’ is the string name and 100 is *Index1*.

Same consideration has to be made for H, HF, HC, L, LS, SN and SC variables. The additional parameters *Index2* and *Index3* must be used for the variables requiring more indexes (for instance, configuration or asset variables of the ToolCenterPoint).

For string or character type variables (LS, SC, HC and HS) the user must define a maximum number of characters to read using *NumCha*.

This function will be available starting from OPENcontrol V3.0.

Return values:

The following table lists all the possible values of *ret_code*:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x00170006	Variable format not consistent with the read variable type
0x00170007	Writing exceeding the variable dimension
0x0017000B	Not existent variable

Example

```
Ret      : dword;
StringVal : STRING ;
DoubleVal : LREAL ;

Ret := Var_ReadEx (1, 100, 0, 0, 10, 'LS', StringVal) ;
Ret := Var_ReadEx (1, 66, 0, 0, 0, 'E', DoubleVal) ;
```

See also

[VAR_WriteEx](#)

2.89 VAR_WriteEx

VAR_WriteEx function writes a process variable.

Syntax

```
ret_code := VAR_WriteEx ( Process, Index1 ,Index2, Index3, NumCha, Name, Value) ;
```

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>Index1</i> (INT)	First index variables
<i>Index2</i> (INT)	Second index variables
<i>Index3</i> (INT)	Third index variables
<i>NumCha</i> (INT)	Number of characters to read for string variables type
<i>Name</i> (STRING)	Variable name
<i>Value</i> (ANY_ELEMENTARY)	Variable value

Possible Overloads

VAR_WriteEx(INT, INT, INT, INT, INT, STRING, BOOL)
VAR_WriteEx (INT, INT, INT, INT, INT, STRING, BYTE)
VAR_WriteEx (INT, INT, INT, INT, INT, STRING, WORD)
VAR_WriteEx (INT, INT, INT, INT, INT, STRING, INT)
VAR_WriteEx (INT, INT, INT, INT, INT, STRING, DWORD)
VAR_WriteEx (INT, INT, INT, INT, INT, STRING, DINT)
VAR_WriteEx (INT, INT, INT, INT, INT, STRING, LREAL)
VAR_WriteEx (INT, INT, INT, INT, INT, STRING, STRING)

Execution mode

Immediate

Use

This function writes a process variable having an input name in the *Name* parameter. This parameter only contains the variable name and its “index”; for instance, in the E100 variable, ‘E’ is the string name and 100 is *Index1*.

Same consideration has to be made for H, HF, HC, L, LS, SN and SC variables. Additional parameters *Index2* and *Index3* must be used for the variables requiring more indexes (for instance, configuration or asset variables of the ToolCenterPoint).

For string or character type variables (LS, SC, HC and HS) the user must define a maximum number of characters to read using *NumCha*.

This function will be available starting from OPENcontrol V3.0.

Return values

The following table lists all the possible values of *ret_code*:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent
0x00170006	Variable format not consistent with the read variable type
0x00170007	Reading exceeding the variable dimension
0x0017000B	Not existent variable

Example

```
Ret      : dword;
StringVal : STRING := 'Variabile LS';
DoubleVal : LREAL := 123.765;

Ret := Var_WriteEx (1, 100, 0, 0, LEN (StringVal), 'LS', StringVal) ;
Ret := Var_WriteEx (1, 66, 0, 0, 0, 'E', DoubleVal) ;
```

See also

[VAR_ReadEx](#)

2.90 AbortTappingCycle

The function AbortTappingCycle aborts the tapping cycle.

Syntax:

```
ret_code := AbortTappingCycle ( Process );
```

Input parameters:

Process (INT) Process number [1..24]

Execution mode:

NoWait

Use:

The command is only accepted during the current tapping cycle (and ignored during the rapid approach to the drill point and during the rapid feed return).

Return values:

The following table lists all the possible values of *ret_code*:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or not existent

Example

```
Ret : dword;  
Ret := AbortTappingCycle (1) ;
```

2.91 SetDynamicAxOverride

SetDynamicAxOverride function selects the Dynamic Override percentage and mode for specified axes.

Syntax

```
ret_code := SetDynamicAxOverride (ExecMode, ExecStatus, Process, Mode,  
AxisId, OverValue) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Mode</i> (BYTE)	Override enabling mode
<i>AxisId</i> (INT)	Axis identifier [1...64]
<i>OverValue</i> (LREAL)	Override reference value

Possible overloads

SetDynamicOverride(INT,DWORD,INT,BYTE,INT,LREAL,[INT,LREAL],...)

Execution mode

Wait / NoWait

Use

If **Mode = pDYN_PERC** (1) the dynamic motion is reduced by the percentage specified, that is the axis feed rate, its acceleration and the jerk will be simultaneously reduced by the percentage value passed (minimum increment 1/100% where 10000=100%). This operation is made only after a motion of the specified axis.

If **Mode = pDYN_VALUE** (2) the dynamic motion is reduced to set the axis feed rate to the same value passed as parameter. In this case, an internal percentage reduction will be calculated to reach the override required. In this case too, acceleration and jerk will be simultaneously reduced by a percentage according to the recalculated value. This operation is done only following the motion of a specified axis.

Override is intended ONLY as dynamic reduction factor and not as an incremental factor, therefore percentage values higher than 100% will be limited to 100%; feed values higher than axis programmed feed, will be limited to the programmed feed.

When an axis is interpolated with other axes, dynamic changes made with the SetDynamicAxOverride function will influence other axes not adjusted by the Dynamic Override.

Return values

The following table lists all possible values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example

```
Ret      : dword;
InterpId : INT;
...
(* Sets % at 50 for axes with ID 1 and 2 on number 1 process *)
Ret := SetDynamicAxOverride (MODE_WAIT, UL0, 1, DYN_PERC,
                           1, 50, 2, 50) ;
```

2.92 ToolCenterPointCFGWrite

The function ToolCenterPointCFGWrite updates the TCP configuration for a given process.

Syntax

```
ret_code := ToolCenterPointCFGWrite (Process, KinId, Config) ;
```

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>KinId</i> (INT)	Kinematics number [1..4]
<i>Config</i> (TCP_StdStruct)	Structure containing the TCP configuration data

Execution mode

Immediate

Use

The function allows the writing of the TCP (Tool Center Point) configuration parameters for the process specified in the *Process* variable and for the kinematics specified by *KinId*. The complete parameter description can be found in the OPENcontrol programming manual. In the structure *Config* (TCP_StdStruct type) the fields below are defined:

General data

Field	Type	Description
<i>KinType</i>	INT	Kinematics type
<i>AbsAxID</i>	INT	Abscissa axis id
<i>OrdAxID</i>	INT	Ordinate axis id
<i>VrtAxID</i>	INT	Vertical axis id
<i>TolAxID</i>	INT	Rotary axis id near the tool
<i>WrkAxID</i>	INT	Rotary axis id near the part
<i>MidAxID</i>	INT	Central rotary axis id
<i>TDirAxNam</i>	STRING[1]	Axis name tool direction (global)
<i>TAscAxNam</i>	STRING[1]	Axis name abscissa tool (global)
<i>TOrdAxNam</i>	STRING[1]	Axis name ordinate tool (global)
<i>CompRadius</i>	INT	Tool radius compensation mode
<i>CompRealtime</i>	INT	Real time compensation tool enabled
<i>DynMode</i>	INT	Dynamic mode (NOT used)
<i>LopMode</i>	WORD	Operative limits management mode
<i>ToolHol</i>	INT	Toolholdernumber enabled
<i>ToolContAng</i>	LREAL	Tool radius contact angle (NOT used)
<i>ToolContEdg</i>	LREAL	Contact angle thoroidal radius(NOT used)
<i>OpRadTolAx</i>	LREAL	Operating radius rotary axis near the tool
<i>OpRadWrkAx</i>	LREAL	Operating radius rotary axis near the part
<i>OpRadMidAx</i>	LREAL	Operating radius central rotary axis
<i>ToolHold</i>	Array of TCP_ThldStruct	Description array of 4 toolholders

Data for prismatic or double twist heads

Field	Type	Description
TolAxFixAng	LREAL	Angle for rotary axis near the tool (if the axis does not exist)
TolAxDirect	LREAL	Rotary axis direction near the tool
WrkAxFixAng	LREAL	Angle for rotary axis near the part (if the axis does not exist)
WrkAxDirect	LREAL	Direction of the rotary axis near the part
WrkAxOffset	LREAL	Offset of the rotary axis near the part
ParamA	LREAL	Distance between Pc and Pr along the first linear axis
ParamB	LREAL	Distance between Pc and Pr along the second linear axis
ParamC	LREAL	Distance between Pc and Pr along the third linear axis
Inclination	LREAL	Rotary axes inclination
ProgMode	LREAL	Special programming mode (NOT used)
NegLimop	LREAL	Lower operating limit (NOT used)
PosLimop	LREAL	Upper operating limit (NOT used)

The information below will be present starting from the OPENcontrol V3.2:

P1Head	Array of LREAL	First point misalignment mounting head
P2Head	Array of LREAL	Second point misalignment mounting head
P3Head	Array of LREAL	Third point misalignment mounting head
ZeroTCP	Array of LREAL	Zero point TCP

Data for HSM heads

Field	Type	Description
MachineType	INT	Machine type
BedType	INT	Machine bed type
ParamABS	Array of LREAL	Kinematics description first linear axis
ParamORD	Array of LREAL	Kinematics description second linear axis
ParamVRT	Array of LREAL	Kinematics description third linear axis
ParamRotT	Array of LREAL	Kinematics description rotary axis near the tool
ParamRotW	Array of LREAL	Kinematics description rotary axis near the part
ParamRotM	Array of LREAL	Kinematics description central rotary axis
ParamWrk	Array of LREAL	Kinematics description part placement

For each of the four configurable toolholders, there is a description structure (TCP_ThldStruct) described as follow:

Data for prismatic or double twist heads

Field	Type	Description
TolAxOffset	LREAL	Roatary axis offset near the tool
ParamD	LREAL	Distance between Pr and Pu along the first linear axis
ParamE	LREAL	Distance between Pr and Pu along the second linear axis
ParamF	LREAL	Distance between Pr and Pu along the third linear axis
Len1Rinv	LREAL	First length for right angle head
Ang1Rinv	LREAL	First angle for right angle head
Len2Rinv	LREAL	Second length for right angle head
Ang2Rinv	LREAL	Second angle for right angle head

Data for HSM heads

Field	Type	Description
ParamTol	Array of LREAL	Tool axis kinematics description
ParamAng	Array of LREAL	Right angle head kinematics description

The following descriptions are valid for some fields:

KinType	Mnemonic	Description
1	TCP_DTWIST	DoubleTwist or MAKAKinematics
2	TCP_VERSOR	Kinematics with versors only
10	TCP_HSM	Kineamtics in HSM mode

CompRad	Mnemonic	Description
0	TCP_NOCOMP	No radius compensation
1	TCP_MNO	Radius compensation with mno verson
2	TCP_MNO_PQD	Radius compensation with mno and pqd versors

LopMode	Mnemonic	Description
0x0001	TCP_LOPDisPRGall	Disables pre-calculation limits
0x0002	TCP_LOPEnaTIPx	Enables realtime limits tool tip first linear axis
0x0004	TCP_LOPEnaTIPy	Enables realtime limits tool tip second linear axis
0x0008	TCP_LOPEnaTIPz	Enables realtime limits tool tip third linear axis
0x0010	TCP_LOPDisPRGx	Disables realtime limits tool tip first linear axis
0x0020	TCP_LOPDisPRGy	Disables realtime limits tool tip second linear axis
0x0040	TCP_LOPDisPRGz	Disables realtime limits tool tip third linear axis

Machine Type	Mnemonic	Description
0	TCP_Mac_TOL	Machine with rotary axes near the tool
1	TCP_Mac_MIX	Machine with two rotary axes: one near the tool and the other near the part
2	TCP_Mac_WRK	Machine with rotary axes near the part
3	TCP_MAC_WC1	Machine with rotary axes near the part + 1 axis sleeve
4	TCP_MAC_WC2	Machine with rotary axes near the part + 2 axis sleeve
5	TCP_MAC_TC1	Machine with rotary axes near the tool + 1 axis sleeve
6	TCP_MAC_TC2	Machine with rotary axes near the tool + 2 axis sleeve
7	TCP_MAC_C12	MIX machine with sleeves
7	TCP_MAC_MIX2	Machine with a rotary axis near the tool and two near the part

BedType	Mnemonic	Description
0	TCP_Bed_WRK	Machine Bed after the part
1	TCP_Bed_B12	Machine Bed after the first sleeve
2	TCP_Bed_BC1	Machine Bed before the second sleeve
3	TCP_Bed_B23	Machine Bed after the second sleeve
4	TCP_Bed_BC2	Machine Bed before the third sleeve
5	TCP_Bed_TOL	Machine Bed before the tool

Return values

The following table lists the values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without error
0x00170005	Process number wrong or non-existent
0x00170018	Wrong command parameter

Example

```
Ret : dword;
Config : TCP_StdStruct;
Ret := ToolCenterPointCFGWrite (1, Config) ;
```

See also

[ToolCenterPointOFF](#), [ToolCenterPointON](#), [ToolCenterPointRestore](#), [ToolCenterPointCFGRead](#)

2.93 ToolCenterPointWrite

The ToolCenterPointWrite function updates the TCP operatin status for a given process.

Syntax

```
ret_code := ToolCenterPointWrite (Process, ControlW, Status) ;
```

Input parameters

<i>Process</i> (INT)	Process number [1..24]
<i>ControlW</i> (INT)	Control Word for writing
<i>Status</i> (TCP_RealtStruct)	Structure containing real-time data

Execution mode

Immediate

Use

The function updates the Tool Centre Point (TCP) status for the process specified in the *Process* variable. In the *Status* structure, TCP_RealtStruct type, some fields to be used for the TCP status update are defined; some are not used (they are present only because the same structure is used in the Read phase) others are used during the update, only if enabled by the Control Word, that is the *ControlW* parameter.

The structure consists of the following fields:

Field	Type	Description
TCPType	INT	TCP type enabled. NOT used
KinId	INT	Kinematics number enabled. NOT used
ToolHld	INT	Tool holders' number enabled. NOT used
TolAxPos	LREAL	Rotary axis position close to the tool
WrkAxPos	LREAL	Rotary axis position close to the part
MidAxPos	LREAL	Central rotary axis position
Vers_i	LREAL	Tool direction versor (i component)
Vers_j	LREAL	Tool direction versor (j component)
Vers_k	LREAL	Tool direction versor (k component)
VarToolLen	LREAL	Tool length change required
VarToolRad	LREAL	Tool radius change required
VarToolEdge	LREAL	Tool toroid radius change required

The Control Word that enables writing is defined as follow:

ControlW	Mask	Field where writing is enabled
0	16#0001	TolAxPos
1	16#0002	WrkAxPos
2	16#0004	MidAxPos
3	16#0008	Vers_i
4	16#0010	Vers_j
5	16#0020	Vers_k
6	16#0040	VarToolLen
7	16#0080	VarToolRad
8	16#0100	VarToolEdge

Return values

The following table lists all possible values that the *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent
0x00170018	Wrong command parameter

Example

```
Ret : dword;
Status : TCP_RealtStruct;
Status.TolAxPos := 0.0 ;
Status.WrkAxPos := 90.0 ;
(* Signals the positions of the first two rotary axes *)
Ret := ToolCenterPointWrite (1, 16#0003, Status) ;
```

See also

[ToolCenterPointOFF](#), [ToolCenterPointON](#), [ToolCenterPointRestore](#), [ToolCenterPointRead](#)

2.94 ToolCenterPointRestore

ToolCenterPointRestore function restores the last TCP enabled for a process.

Syntax

ret_code := ToolCenterPointRestore (*Process*, *TcpType*, *KinId*, *RotarySts*) ;

Input parameter

Process (INT) Process number [1..24]

Output parameters

<i>TcpType</i> (INT)	Enabling mode 1 or 5 (for tool direction)
<i>KinId</i> (INT)	Kinematics number [1..4]
<i>RotarySts</i> (INT)	Rotary axes homing status

Execution

Wait

Use

The function allows restoration of the last Tool Centre Point (TCP) enabled for the process specified in the *Process* variable. With this function, only the \$ variables of the TCP configuration will be reloaded (the configuration is in HSM). TCP can be enabled through another call to **ToolCenterPointON**.

In the *TcpType* variable type, returns the status that enabled the TCP and will count values reported in the following table:

TcpType	Mnemonic	Description
1	TCP_NORMAL	TCP without additional axes
5	TCP_TOOLDIR	TCP enabled with tool direction axis
15	TCP_TOOLDIR_3D	TCP with tool axis direction and additional axes integral with the orientation of the head
25	TCP_TOOLDIR_2D	TCP with tool axis direction and additional axes integral with the XY plane of the machine

The *KinId* parameter will return the number of the kinematic used, while the *RotarySts* parameter will return the homing status of the two rotary axes with the following coding

RotarySts Bit	Mask	Description
0	16#0001	Homing on the first rotary axis (Tol)
1	16#0002	Homing on the second rotary axis (Wrk)
2	16#0004	Homing on the third rotary axis (Mid)

Return values

The following table lists all possible values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number
0x00170018	Wrong command parameter

Example

```
Ret : dword;
TcpType : int ;
KinId : int ;
RotarySts : word ;
Ret := ToolCenterPointRestore (1, TcpType, KinId, RotarySts) ;
```

See also

[ToolCenterPointOFF](#), [ToolCenterPointON](#), [ToolCenterPointStatus](#)

2.95 ToolCenterPointRead

ToolCenterPointRead function restores the TCP operating status for a process.

Syntax

```
ret_code := ToolCenterPointRead (Process, Status) ;
```

Input parameters

Process (INT) Process number [1..24]

Output parameters

Status (TCP_RealtStruct) Structure containing real-time data

Execution mode

Immediate

Use

The function determines the Tool Centre Point (TCP) status for the process specified in the *Process* variable. In the *Status* structure (TCP_RealtStruct type) the following fields are always defined:

Field	Type	Description
TCPTYPE	INT	TCP type active
KinId	INT	Number of active kinematics
ToolHld	INT	Number of active tool holders
TolAxPos	LREAL	Rotary axis position close to the tool
WrkAxPos	LREAL	Rotary axis position close to the part
MidAxPos	LREAL	Central rotary axis position
Vers_i	LREAL	Tool direction versor (i component)
Vers_j	LREAL	Tool direction versor (j component)
Vers_k	LREAL	Tool direction versor (k component)
VarToolLen	LREAL	Change in tool length required
VarToolRad	LREAL	Change in tool radius required
VarToolEdge	LREAL	Change in tool toroid radius required

TcpType	Mnemonic	Description
0	TCP_OFF	Inactive TCP
1	TCP_NORMAL	TCP enabled without additional axes
5	TCP_TOOLDIR	TCP enabled with tool direction axis
15	TCP_TOOLDIR_3D	TCP with tool axis direction and additional axes integral with the orientation of the head
25	TCP_TOOLDIR_2D	TCP with tool axis direction and additional axes integral with the XY plane of the machine

Return values

The following table lists all possible values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number
0x00170018	Wrong command parameter

Example

```
Ret : dword;
Status : TCP_RealtStruct;
Ret := ToolCenterPointRead (1, Status) ;
```

See also

[ToolCenterPointOFF](#), [ToolCenterPointON](#), [ToolCenterPointRestore](#), [ToolCenterPointWrite](#)

2.96 ToolCenterPointCFGRead

The ToolCenterPointCFGRead function reads a TCP process configuration.

Syntax

```
ret_code := ToolCenterPointCFGRead (Process, Config) ;
```

Input parameters

<i>Process</i> (INT)	process number [1..24]
<i>KinId</i> (INT)	Kinematics number [1..4]

Output parameters

<i>Config</i> (TCP_StdStruct)	Structure containing TCP configuration data
-------------------------------	---

Execution mode

Immediate

Use

The function reads TCP configuration parameters for the process specified by *Process* variable and for the kinematic specified by *KinId*. The complete description of these parameters is defined in the OPENcontrol programming manual.

In the *Config* structure, TCP_StdStruct type, the following fields are defined:

General data

Field	Type	Description
<i>KinType</i>	INT	Kinematics type
<i>AbsAxID</i>	INT	Abscissa axis id
<i>OrdAxID</i>	INT	Ordinate axis id
<i>VrtAxID</i>	INT	Vertical axis id
<i>TolAxID</i>	INT	Rotary axis id close to the tool
<i>WrkAxID</i>	INT	Rotary axis id close to the part
<i>MidAxID</i>	INT	Central rotary axis id
<i>TDirAxNam</i>	STRING[1]	Tool direction axis name (global)
<i>TAscAxNam</i>	STRING[1]	Tool abscissa axis name (global)
<i>TOrdAxNam</i>	STRING[1]	Tool ordinate axis name (global)
<i>CompRadius</i>	INT	Tool radius offset mode
<i>CompRealtime</i>	INT	Tool real-time offset activation
<i>DynMode</i>	INT	Dynamic mode (NOT used)
<i>LopMode</i>	WORD	Operational limits management mode
<i>ToolHol</i>	INT	Number of tool holders active
<i>ToolContAng</i>	LREAL	Tool radius contact angle (NOT used)
<i>ToolContEdg</i>	LREAL	Toroid radius contact angle (NOT used)
<i>OpRadTolAx</i>	LREAL	Operating radius rotary axis close to the tool
<i>OpRadWrkAx</i>	LREAL	Operating radius rotary axis close to the part
<i>OpRadMidAx</i>	LREAL	Operating radius central rotary axis
<i>ToolHold</i>	Array of TCP_ThldStruct	4 tool holders array description

Double twist or prismatic heads data

Field	Type	Description
TolAxFixAng	LREAL	Rotary axis angle close to the tool (if the axis does not exist)
TolAxDirect	LREAL	Rotary axis direction close to the tool
WrkAxFixAng	LREAL	Rotary axis angle close to the part (if the axis does not exist)
WrkAxDirect	LREAL	Rotary axis direction close to the tool
WrkAxOffset	LREAL	Rotary axis offset close to the part
ParamA	LREAL	Distance on the first linear axis between Pc and Pr
ParamB	LREAL	Distance on the second linear axis between Pc and Pr
ParamC	LREAL	Distance on the third linear axis between Pc and Pr
Inclination	LREAL	Inclination among rotary axes
ProgMode	LREAL	Special programming mode (NOT used)
NegLimop	LREAL	Negative operative limit (NOT used)
PosLimop	LREAL	Positive operative limit (NOT used)

From V3.2 of OPENcontrol the following information will be available:

P1Head	Array of LREAL	First point misalignment mounting head
P2Head	Array of LREAL	Second point misalignment mounting head
P3Head	Array of LREAL	Third point misalignment mounting head
ZeroTCP	Array of LREAL	Zero TCP point

HSM heads data

Campo	Type	Description
MachineType	INT	Machine type
BedType	INT	Bed machine type
ParamABS	Array of LREAL	First linear axis kinematic description
ParamORD	Array of LREAL	Second linear axis kinematic description
ParamVRT	Array of LREAL	Third linear axis kinematic description
ParamRotT	Array of LREAL	Rotary axis close to the tool kinematic description
ParamRotW	Array of LREAL	Rotary axis close to the part kinematic description
ParamRotM	Array of LREAL	Rotary central axis kinematic description
ParamWrk	Array of LREAL	Part positioning kinematic description

For each of the 4 tool holders there is a TCP_ThldStruct structure description as follow:

Double twist or prismatic heads data

Field	Type	Description
TolAxOffset	LREAL	Rotary axis offset close to the tool
ParamD	LREAL	Distance on the first linear axis between Pr and Pu
ParamE	LREAL	Distance on the second linear axis between Pr and Pu
ParamF	LREAL	Distance on the third linear axis between Pr and Pu
Len1Rinv	LREAL	First length for right angle head
Ang1Rinv	LREAL	First angle for right angle head
Len2Rinv	LREAL	Second length for right angle head
Ang2Rinv	LREAL	Second angle for right angle head

Data for HSM heads

Field	Type	Description
ParamTol	Array of LREAL	Tool axis kinematic description
ParamAng	Array of LREAL	Right angle head kinematic description

For some fields the followi descriptions are valid

KinType	Mnemonic	Description
1	TCP_DTWIST	DoubleTwist or MAKAKinematics
2	TCP_VERSOR	Kinematic with versors only
10	TCP_HSM	Kinematic in HMS mode

CompRad	Mnemonic	Description
0	TCP_NOCOMP	No radius compensation
1	TCP_MNO	Radius compensation with mno verson
2	TCP_MNO_PQD	Radius compensation with mno and pqd versors

LopMode	Mnemonic	Description
0x0001	TCP_LOPDisPRGall	Disables pre-calculation limits
0x0002	TCP_LOPEnaTIPx	Enables tool tip real-time limits for the first linear axis
0x0004	TCP_LOPEnaTIPy	Enables tool tip real-time limits for the second linear axis
0x0008	TCP_LOPEnaTIPz	Enables tool tip real-time limits for the third linear axis
0x0010	TCP_LOPDisPRGx	Disables pre-calculation limits for the first linear axis
0x0020	TCP_LOPDisPRGx	Disables pre-calculation limits for the second linear axis
0x0040	TCP_LOPDisPRGz	Disables pre-calculation limits for the third linear axis

MachineType	Mnemonic	Description
0	TCP_Mac_TOL	Machine with rotary axis close to the tool
1	TCP_Mac_MIX	Machine with two rotary axes, one close the tool and the other close to the part
2	TCP_Mac_WRK	Machine with rotary axes close to the part
3	TCP_MAC_WC1	Machine with rotary axes close to the part + sleeve 1 axis
4	TCP_MAC_WC2	Machine with rotary axes close to the part + sleeve 2 axes
5	TCP_MAC_TC1	Machine with rotary axes close to the tool + sleeve 1 axis
6	TCP_MAC_TC2	Machine with rotary axes close to the tool + sleeve 2 axes
7	TCP_MAC_C12	MIX machine with sleeves
8	TCP_Mac_MIX2	Machine with one rotary axis close to the tool and two others close to the part

BedType	Mnemonic	Description
0	TCP_Bed_WRK	Machine Bed after the part
1	TCP_Bed_B12	Machine Bed after the first sleeve
2	TCP_Bed_BC1	Machine Bed before the second sleeve
3	TCP_Bed_B23	Machine Bed after the second sleeve
4	TCP_Bed_BC2	Machine Bed before the third sleeve
5	TCP_Bed_TOL	Machine Bed before the tool

Return values

The following table lists all possible values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number
0x00170018	Wrong command parameter

Example

```
Ret : dword;
Config : TCP_StdStruct;
Ret := ToolCenterPointCFGRead (1, Config) ;
```

See also

[ToolCenterPointOFF](#), [ToolCenterPointON](#), [ToolCenterPointRestore](#), [ToolCenterPointCFGWrite](#)

2.97 ToolCenterPointON

ToolCenterPointON function enables TCP use for a process.

Syntax

```
ret_code := ToolCenterPointON (ExecMode, ExecStatust, Process, TcpType, KinId) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>TcpType</i> (INT)	Enabling mode 1 or 5 (for tool direction)
<i>KinId</i> (INT)	Kinematics number [1..4]

Execution mode

Wait / NoWait

Use

The function enables the Tool Centre Point (TCP) for the process specified in the *Process* variable. Through the *TcpType* variable, the TCP functionality is defined according to the following table:

TcpType	Mnemonic	Description
1	TCP_NORMAL	TCP enabled without additional axes
5	TCP_TOOLDIR	TCP enabled with tool direction axis
15	TCP_TOOLDIR_3D	TCP with tool axis direction and additional axes integral with the orientation of the head
25	TCP_TOOLDIR_2D	TCP with tool axis direction and additional axes integral with the XY plane of the machine

The *KinId* parameter identifies the number of kinematics to enable. All kinematic characterization variables must have been uploaded either through ODM or PartProgram. The process must be in IDLE status.

Return values

The following table lists all possible values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number
0x00170018	Wrong command parameter

Example

```
Ret : dword;
Ret := ToolCenterPointON (MODE_WAIT, MLO, 1, TCP_TOOLDIR, 1) ;
```

See also:

ToolCenterPointOFF, ToolCenterPointStatus, ToolCenterPointRestore

2.98 ToolCenterPointOFF

The function ToolCenterPointOFF disables TCP for a given process.

Syntax

```
ret_code := ToolCenterPointOFF (ExecMode, ExecStatust, Process) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]

Execution mode

Wait / NoWait

Use

The function disables Tool Center Point (TCP) for the process specified by the *Process* variable. The process must be in IDLE status.

Return values

The following table lists all the values the variable *ret_code* can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent
0x00170018	Wrong command parameter

Example

```
Ret : dword;
Ret := ToolCenterPointOFF (MODE_WAIT, ML0, 1) ;
```

See also

[ToolCenterPointON](#), [ToolCenterPointOFF](#), [ToolCenterPointRestore](#)

2.99 ToolCenterPointStatus

ToolCenterPointStatus function restores the TCP operating status for a process.

Syntax

```
ret_code := ToolCenterPointStatus (Process, TcpType, KinId, RotarySts) ;
```

Input parameter

Process (INT) Process number [1..24]

Output parameters

<i>TcpType</i> (INT)	Enabling mode 1 or 5 (for tool direction)
<i>KinId</i> (INT)	Kinematics number [1..4]
<i>RotarySts</i> (INT)	Rotary axes homing status

Execution mode

Wait

Use

The function knows the Tool Centre Point (TCP) status for the process specified in the *Process* variable. In the *TcpType* variable the TCP activation status will be returned according to the following table:

TcpType	Mnemonic	Description
0	TCP_OFF	Inactive TCP
1	TCP_NORMAL	TCP enabled without additional axes
5	TCP_TOOLDIR	TCP enabled with tool direction axis
15	TCP_TOOLDIR_3D	TCP with tool axis direction and additional axes integral with the orientation of the head
25	TCP_TOOLDIR_2D	TCP with tool axis direction and additional axes integral with the XY plane of the machine

The *KinId* parameter will return the number of the enabled kinematic, while the *RotarySts* parameter will return the homing status of the two rotary axes with the following codes:

RotarySts Bit	Mask	Description
0	16#0001	Homing executed on the first rotary axis (Tol)
1	16#0002	Homing executed on the second rotary axis (Wrk)
2	16#0004	Homing executed on the third rotary axis (Mid)

Return values

The following table lists all possible values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number
0x00170018	Wrong command parameter

Example

```
Ret : dword;
TcpType : int ;
KinId : int ;
RotarySts : word ;
Ret := ToolCenterPointStatus (1, TcpType, KinId, RotarySts) ;
```

See also

[ToolCenterPointOFF](#), [ToolCenterPointON](#), [ToolCenterPointRestore](#)

2.100 SetDynamicOverride

SetDynamicOverride function selects Dynamic Override percentage and mode.

Syntax

```
ret_code := SetDynamicOverride (ExecMode, ExecStatust, Process, Mode, OverValue) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]
<i>Mode</i> (BYTE)	Override enabling mode
<i>OverValue</i> (LREAL)	Override reference value

Execution mode

Wait / NoWait

Use

This function is used for a real-time monitoring of the axes feedrate. The dynamic is controlled according to the Mode parameter.

With **Mode = pDYN_CLEAR** (0) the Dynamic Override function is disabled; it is like programming a 100% override.

In case of **Mode = pDYN_PERC** (1)there is a percentage dynamic reduction, that is the feed rate, the acceleration and the jerk will be simultaneously reduced by the percentage value passed (minimum increment 1/100 - that is 10000 = 100%).

Mode = pDYN_VALUE

In case of **Mode = pDYN_VALUE** (2) dynamic is reduced in order to set the feedrate to the same value as that passed as parameter. In this case it will be calculated an internal percentage reduction to perform the requested override. Also in this case the jerk and the acceleration will be simultaneously reduced by a percentage according to the recalculated value. The override is ONLY a dynamic reduction factor and not an incremental factor, therefore percentage values higher than 100% will be limited to 100%, while feed values higher than the actual feed, will be limited to the actual feed.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Wrong or non-existent process number

Example

```
Ret      : dword;
Dummy   : dword;
Ret := SetDynamicOverride (MODE_NOWAIT, Dummy, 1, pDYN_PERC, 10000.0) ;
```

2.101 Refresh3DPositions

The Refresh3DPositions function refreshes the 3D axes positions in a process.

Syntax

```
ret_code := Refresh3DPositions (ExecMode, ExecStatus, Process) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Process</i> (INT)	Process number [1..24]

Execution mode

Wait / NoWait

Use

This function must be used if there is a change in the position of the process axes involved in TCP or UPR functions or various virtualizations. PLC can change the position without notifying the process (for example ZeroShift motion) or by disabling and re-enabling the axes. Following these operations the process must be informed to realign the known positions.

If the call occurs during HOLD status, before restoring the program execution, it will be necessary to run return on the profile.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00170005	Process number wrong or non-existent

Example

```
Ret      : dword;
Dummy   : dword;

Ret := Refresh3DPositions (MODE_NOWAIT, Dummy, 1) ;
```

2.102 CamAxiDefine

The function CamAxiDefine defines an electronic cam associated with a master and to a slave axis.

Syntax

```
ret_code := CamAxiDefine (Process, MasterId, SlaveId, FileName, out CamId) ;
```

Input parameters

<i>Process</i> (INT)	Process identifier [1..24]
<i>MasterId</i> (INT)	Master axis identifier [1..64]
<i>SlaveId</i> (INT)	Slave axis identifier [1..64]
<i>FileName</i> (STRING)	String containing the correspondence between master and slave

Output parameter

<i>CamId</i> (INT)	Electronic cam identifier [1..64]
--------------------	-----------------------------------

Execution mode

Wait

Use

The function CamAxiDefine builds an association between a slave axis and a master. This association is predefined in tables in the form of positions that the slave must assume according to the position of the master. It is possible to create several electronic cams associated with the same master and slave, just by changing the input file.

Tables are created from a file in .csv format. The file is organized as strings, where each string refers to a cam position. The first string contains the information referring to the file type and the number of compiled strings. There can be two types of files:

A file with two columns where the first contains the positions of the master and the second the corresponding positions of the slave.

First string →	0;4	FileType; Number of Strings
	100;10	M;S
	150;13	
	200;18	
	250;10	

A file with multiple columns of which the first contains the positions of the master and the subsequent coefficients of polynomials from which the corresponding positions of the slave are created.

First string →	1;4	FileType; Number of Strings
	100;1;0.2;3.3;4;5	M;pS0;pS1;pS2;pS3;pS4;pS5
	150;;;;;	
	200;;;;;	
	250;;;;;	

When the cam is enabled, the master position in the table is searched for and the slave axis position is calculated by linearly interpolating (or via polynomials) the two points including the master position. The master axis positions must be an ordered set of increasing or decreasing polynomials.

Return values

The following table lists all the values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0016002C	Axis ID not accepted
0x0016002B	Axis ID not found
0x0016002D	Axis ID referring to a gantry
0x00160027	Axis ID referring to a spindle
0x0016002A	Axes with different clocks
0x00160146	Reached the maximum number of cams
0x00160140	Error opening the file
0x00160141	LF not found at the end of the file
0x00160142	File non-existent
0x00160143	Table not ordered
0x00160144	First line wrong format
0x00160145	Wrong string format

Example

```

Ret      : dword;
CamId   : int ;
FileName : string := '\SSD\OSAI\Camma.csv';

0;9
0;0
100;100
200;200
300;300
400;400
500;300
600;200
700;100
800;0

(*Creates an electronic cam between the master axis having Id 1 and the slave
with Id 2*)
Ret := CamAxiDefine(Process,1,2,FileName, CamId);

```

See also

[CamAxiStart](#), [CamStop](#), [CamUndefine](#), [GMC_ReadAxisInfo](#)

2.103 CamAxiStart

The function CamAxiStart enables one or more master/slave type electronic cams.

Syntax

```
ret_code :=GMC_CamAxiStart(Process, OffsetMaster, Mode, Rollover, OffsetSlave, CamStr);
```

Input parameters

<i>Process(INT)</i>	Process identifier[1..24]
<i>OffsetMaster(LREAL)</i>	Offset value (early or late) on the application of the slave position
<i>Mode(INT)</i>	Following mode
<i>Rollover(INT)</i>	Cyclic cam modulus
<i>OffsetSlave(LREAL)</i>	Offset value applied to the slave position

Input/output parameters

<i>CamStr (CAMCNC_Struct)</i>	Structure containing the id of the cam to be enabled and the id of the action to which is associated after the enabling.
-------------------------------	--

Possible overloads

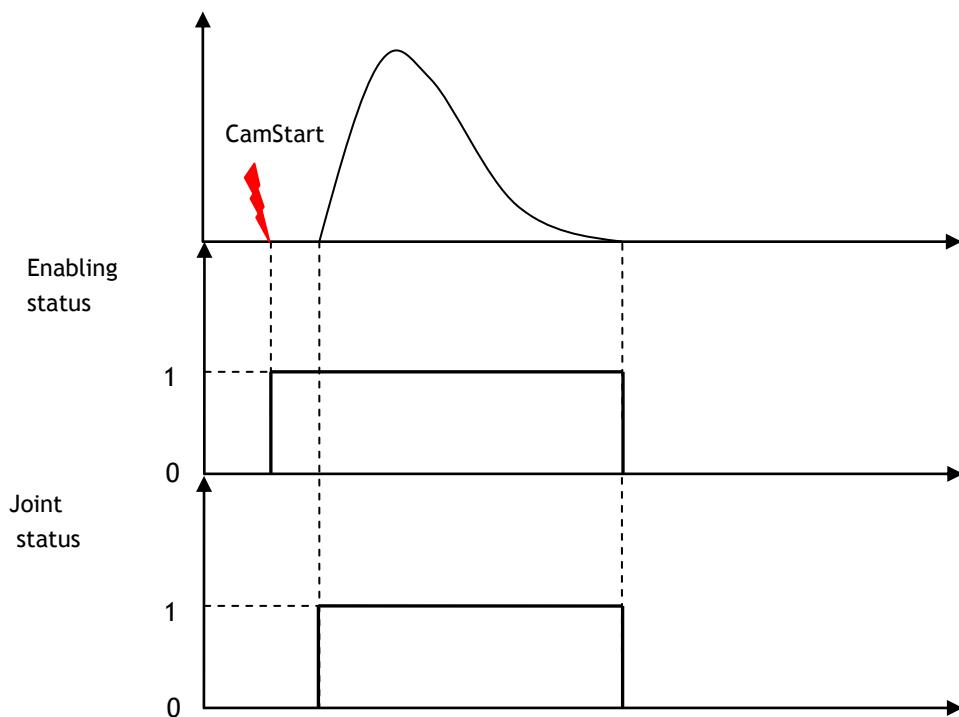
```
CamAxiStart(Process ,OffsetMaster, Mode, Rollover,[OffsetSlave, CamStr]...);
```

Execution mode

Wait

Use

The function CamAxiStart enables the entered electronic cams, referring to each of them an “ActionId”. The activation is immediate but the coupling of the slave depends on the position of the master. The figure below explains this concept better:



The real-time software searches in the table for the master position and, when it is found, calculates the point to pass to the slave. During the activation, the slave may not be close to the first point of the cam profile, therefore, the slave must move with its best dynamic features to reach the required position. According to the active following mode, the error between the actual and Theoretical position could be recovered from the axis during the ongoing movement, if possible, at speeds greater than that programmed.

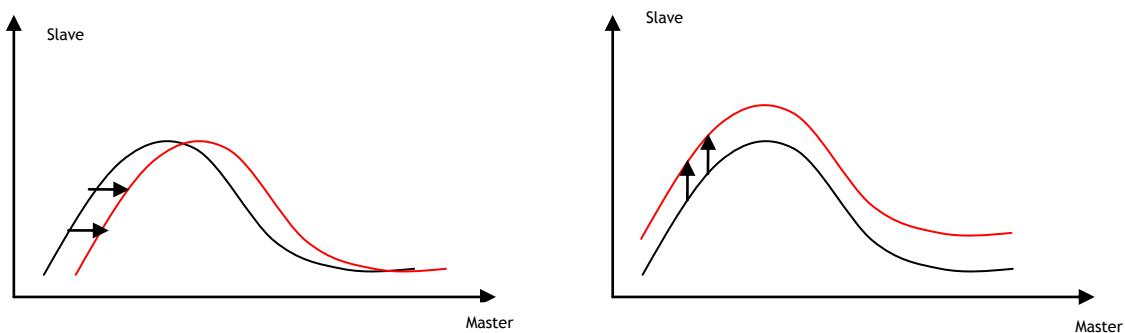
The engagement and disengagement positions are, respectively, the first and last points of the cam table.

The position of the master found in the table is given by the following relationship:

$$Pm = Pm_{att} + OffsetMaster$$

where Pm_{att} is the master position including origins (in case it is an interpolated axis).

Figures below shows the effect of *OffsetMaster* and *OffsetSlave* applications on the cam profile:



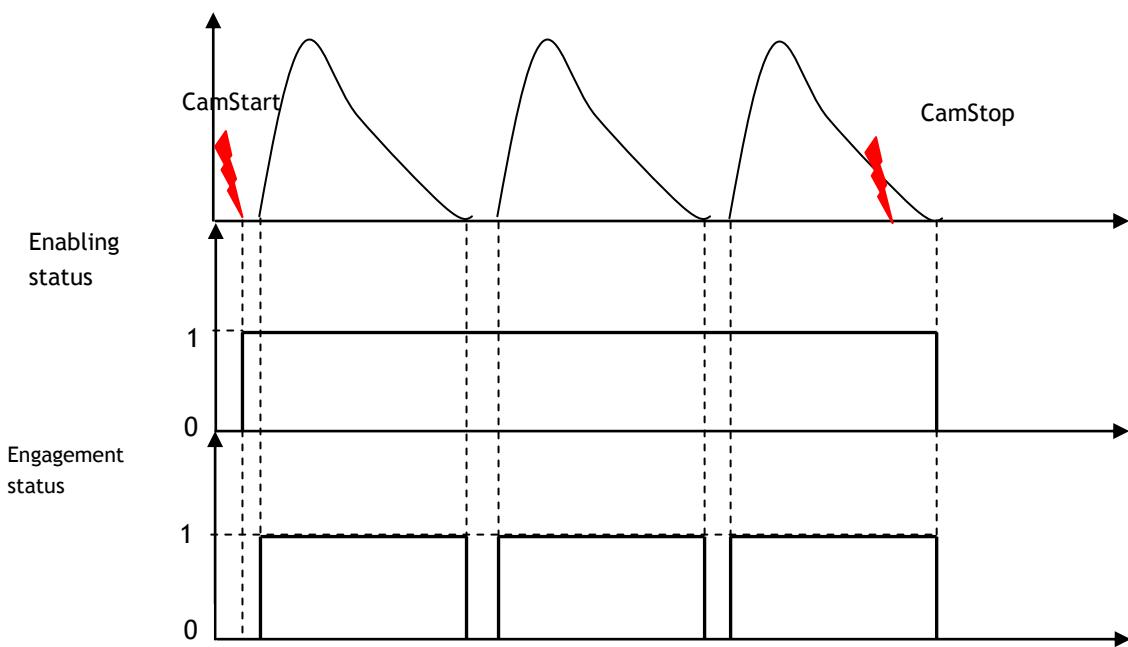
Description of the structure *CamCNC_Struct*:

Field	Type	Description
CamID	INT	Identifier of the cam to enable
ActionId	INT	Identifier of the action "Action" associated to the cam enabled[1...128]

Description of the file *Mode*:

Mode	Mnemonic	Description
0	CAM_INT	Electronic CAM standard interpolation mode
1	CAM_AXF	Electronic CAM with AXF tracking mode

The input Rollover determines if the cam has to be done once (Rollover = 0) or indefinitely (Rollover = cyclic form of the cam) until an explicit stop command. The following figure shows an example of cyclic cam:



Return values

The following table lists all the values *ret_code* variable can have:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160147	Maximum number of Action reached
0x00160148	Invalid cam ID
0x0016014A	Undefined cam
0x0016014B	Invalid cyclic cam modulus (in case of real rollover axis with rate lower than the cam modulus)
0x0016014D	Release position higher than the cyclic cam
0x00160028	Axis in use from another interpolator
0x00160029	Axis in use from another interpolator in Hold

Example

```

Ret      : dword;
CamId   : int ;
CamStr1 : CamCNC_Struct;
CamStr2 : CamCNC_Struct;

(*Enables in one-shot the electronic cams having Id 1 and 2 with OffsetMaster
null and OffsetSlave null fro the first cam and 10.0 for the second. *)
CamStr1.CamId := 1;
CamStr2.CamId := 2;
Ret := CamAxiStart(InterpId, 0.0, 0, 0.0, 0.0, CamStr1, 10.0, CamStr2);

```

See also

[CamAxiDefine](#), [CamStop](#), [CamUndefine](#), [GMC_ReadAxisInfo](#)

2.104 CamOutDefine

The function CamOutDefine defines an electronic cam associated to a master an to some inputs.

Syntax

```
ret_code := CamOutDefine (Process, MasterId, FileName, out CamId) ;
```

Input parameters

Process(INT) Process identifier [1..24]

MasterId(INT) Master identifier

FileName(STRING) String containing the correspondence between master and output

Output parameters

CamId (INT) Electronic cam identifier [1..64]

Execution mode

Wait

Use

The function CamOutDefine creates an association between a master and a serie of outputs that will be set/reset in correspondence with determined master postions. This association is defined in Tables.

Tables are created with a file in .csv format. The file is arranged as strings, therefore each string refers to a cam position. The first string contains information referring to the file type and to the number of compiled strings.

First string →	2;2	FileType; Number of strings
	100;OW(12,0xF000,0xFF00)	M;OW(index, value, mask) or
	200;OW(128,0xF000,0xFF00)	OF(mask, value)

When the cam is enabled, the position of the master is searched in the table and the outputs associated to the corresponding word are set/reset. The positions of the master axis are strictly increasing or decgreasing monotone.

Return values

The following table lists all values the variable that *ret_code* can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0016002C	Axis ID not accepted
0x0016002B	Axis ID not found
0x0016002D	Axis ID referred to a gantry
0x00160027	Axis ID referred to a spindle
0x0016002A	Axes with different clocks
0x00160146	Reached the maximum number of cams
0x00160140	Opening file error
0x00160141	LF not found at the end of file
0x00160142	Non-existent file

0x00160143	Table not strictly monotone
0x00160144	First string format is wrong
0x00160145	String format is wrong

Example

```

Ret      : dword;
CamId    : int ;
FileName : string := '\SSD\OSAI\CammaOUT.csv';

0;9
2;12
10;OW(13,61440,65280)
20;OW(14,0xF000,0xFF00)
30;OW(13,3840,65280)
40;OW(14,0x0F00,0xFF00)
50;OW(13,61440,65280)
60;OW(14,0xF000,0xFF00)
70;OW(13,3840,65280)
80;OW(14,0x0F00,0xFF00)
90;OW(13,61440,65280)
100;OW(14,0xF000,0xFF00)
110;OW(13,3840,65280)
120;OW(14,0x0F00,0xFF00)
(*Creates an electronic cam managing the outputs*)
Ret := CamOutDefine(Process,1,FileName, CamId);

```

See also

[CamOutStart](#), [CamStop](#), [CamUndefine](#), [GMC_ReadAxisInfo](#)

2.105 CamOutStart

The function CamOutStart enables the type of electronic cam output.

Syntax

```
ret_code := CamOutStart(Process, CamID, OffsetMaster, Rollover, ActionID);
```

Input parameters

<i>Process</i> (INT)	Process identifier [1..24]
<i>CamID</i> (INT)	Identifier of the cam to enable [1..64]
<i>OffsetMaster</i> (LREAL)	Offset value (early or late) on the outputs set/reset.
<i>Rollover</i> (INT)	Cyclic cam modulus

Output parameter

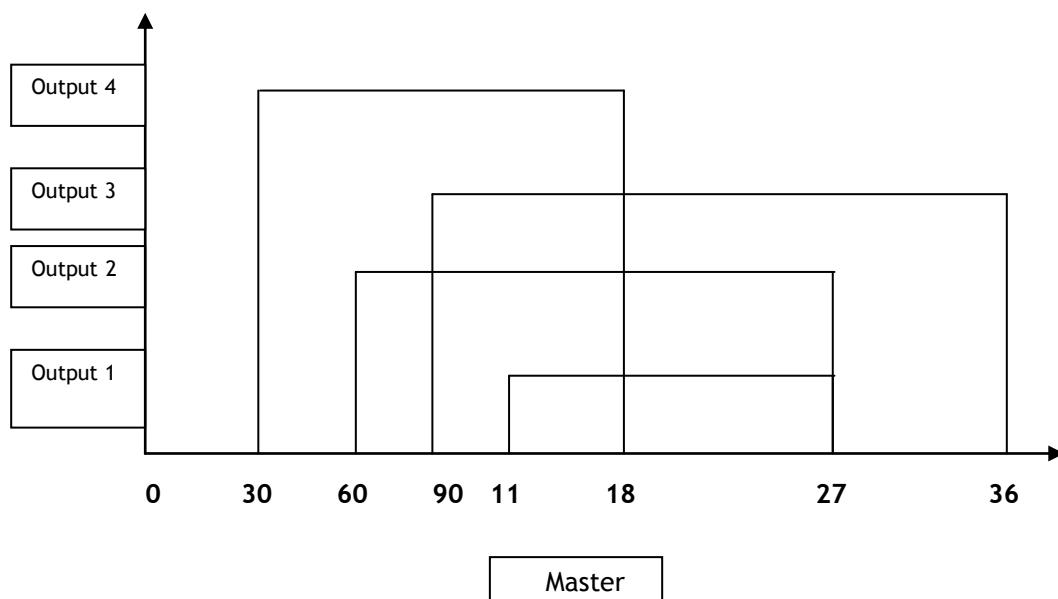
<i>ActionID</i> (INT)	Identifier of the action created [1..128]
-----------------------	---

Execution mode

Wait

Use

The function CamOutStart enables the electronic cam in input associating it with an “ActionId”. The activation is immediate, but the outputs management depends on the master position.



The real-time software finds the position of the master in the table and, when it is found, manages the corresponding outputs.

The master position searched in the table is given by the following relationship

$$Pm = Pm_{att} + OffsetMaster$$

where Pm_{att} is the position of the maser including any origin (in case this is an interpolated axis).

The *Rollover* input determines if the cam has to be followed only once (Rollover=0) or continuously (Rollover = cyclic cam modulus) until an explicit stop command

Return values

The following table lists all values that the *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function execute without errors
0x00160147	Reached the maximum number of Actions
0x00160148	Invalid cam ID
0x0016014A	Undefined cam
0x0016014B	Invalid cyclic cam modulus (in the case of a real rollover axis with a pitch lower than the cam modulus)
0x0016014D	Release position higher than the modulus of the cyclic cam

Example

```
Ret      : dword;
CamId   : int ;
ActionId : int ;
(*Enables in cyclic mode the electronic cam having Id = 1*)
CamId := 1;
Ret := CamOutStart(Process, 1, 360.0, 0, ActionId);
```

See also

[CamOutDefine](#), [CamStop](#), [CamUndefine](#), [GMC_ReadAxisInfo](#)

2.106 CamStop

The function CamStop disables one or more active cams.

Syntax

```
ret_code := CamStop(Process , Mode, ActionId);
```

Input parameters

<i>Process</i> (INT)	Process identifier [1..24]
<i>Mode</i> (INT)	Interruption mode
<i>ActionId</i> (INT)	Identifier of the action to stop [1..128]

Possible overloads

CamStop(*Process*, *Mode*, [*ActionId*]...);

Execution mode

Wait

Use

The function CamStop immediately dis the requested action. The input *Mode* has a different meaning according to the type of cam to stop.

<i>Mode</i>	<i>Mnemonic</i>	Master/Slave cam	Output cam
0	CAM_STOP0	The slave axis immediately stops with its deceleration ramp and the input action is deleted.	Deletes the current action leaving unchanged the outputs status
1	CAM_STOP1	The slave axis only stops after completing the profile. When the movement finishes, the action is deleted. This command is redundant for the one-shot cams because the actions are automatically deleted at the end of the profile execution.	Deletes the current action only after resetting all the outputs

Return values

The following table lists all values the *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160149	Invalid action ID

Example

```
Ret      : dword;
ActionId : int := 1;
(*Immediately disables the electronic cam associated to the action with Id = 1*)
Ret := CamStop(Process, 0, ActionId);
```

See also

CamAxiDefine, CamOutDefine, CamAxiStart, CamOutStart, CamUndefine, GMC_ReadAxisInfo

2.107 CamUndefine

The function CamUndefine deletes one or more input cams.

Sintassi

```
ret_code := CamUndefine(Process, CamId);
```

Input parameters

<i>Process</i> (INT)	Process identifier [1..24]
<i>CamId</i> (INT)	Identifier of the cam to delete [1..64]

Possible overloads

```
CamUndefine(Process, [CamId]...);
```

Execution mode

Wait

Use

The function CamUnDefine deletes the input cams.

Return values

The following table lists all values the *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160148	Invalid cam ID
0x0016014C	Cam busy in an action

Example

```
Ret      : dword;
CamId    : int := 1;
(*Deletes the electronic cam with Id = 1*)
Ret := CamUndefine(Process, 1);
```

See also

[CamAxiDefine](#), [CamOutDefine](#), [CamAxiStart](#), [CamOutStart](#), [CamStop](#), [GMC_ReadAxisInfo](#)

2.108 CNC error list (processes)

Code (HEX)	Description
0x00170001	Syntax Error Syntax error found in the part program block or in the MDI block
0x00170002	Open file NMC_BINARY.DAT error The machine configuration file generated by ODM has not been found. Recompile the AMP and reboot.
0x00170003	Open file LDIRDATA.DAT error The default directory configuration file generated by ODM has not been found. Recompile the AMP and reboot.
0x00170004	Insufficient memory on process Memory full. Reduce the number of variables configured in AMP, or increase available RAM memory.
0x00170005	Process undefined The command applies to a process that has not been configured, or it is a value smaller than 1 or greater than 24
0x00170006	Format error This error is displayed in the following cases: <ul style="list-style-type: none"> ➢ Wrong variable format ➢ Invalid IBSIZE values (allowed values are 0,3,4,5) ➢ Wrong variable index ➢ Feed rate (F) = 0 or negative ➢ Wrong variable format ➢ Repeat number is illegal (number of repetitions must be from 1 to 65535) ➢ Format error in assignment, e.g. assignment to strings with different lengths ➢ PLC variables writing/reading error ➢ Character variable format error in DIS code: not specified as CHAR ➢ Protected area not allowed: 0 < protected area number < 4 ➢ Variable not configured ➢ Wrong parameter for a command
0x00170007	Wrong variable identifier The index identifying the variable in read/write or assign commands is wrong.

Code (HEX)	Description
0x00170008	<p>Wrong axis name The name of the axis specified in the command is not correct. This error is displayed in the following cases:</p> <ul style="list-style-type: none"> ➢ the axis specified in axis data request functions has not been configured ➢ error in reading axis position, because specified axis does not exist ➢ the axis specified in library calls CncCalcAxOfs, CncAxOfsAct, CncCalcToolOfs, CncToolOfsAct, CncUp, CncSelAxi does not exist ➢ the axis programmed during origin management (pre-setting, UIO, UAO, RQO, etc.) or offset pre-setting has not been configured ➢ the axis specified in the triliterals SCF, MIR has not been configured ➢ the definition of the work plan with G17, G18, G19, G16 is wrong, because the number of axes configured is less than 2 or because the axis specified has not been configured ➢ the name of the axis to be selected, homed, shared with logic or blocked has not been found in the table of the axes concerning that process or has not been configured ➢ the axis specified in the triliterals SOL, DPA, AXO, UVW, FLT has not been configured or has been duplicated. ➢ the axis specified in the commands G92, G98, has not been configured
0x00170009	<p>Too many variables in RD/WR The number of variables in read/write commands exceeds maximum limit.</p>
0x0017000A	<p>Wrong axes number in the command The number of axes specified in the command is not correct. Error is displayed to indicate that the number of axes specified in library command CncGetAxShare/CncRelAxShare is higher than the allowed number.</p>
0x0017000B	<p>Read/write error value variable The name of the variable, specified in the library read/write commands, is wrong, or the variable has not been configured</p>
0x0017000C	<p>Wrong data into ODM configuration file Wrong data error.</p>
0x00170010	<p>Command unknown Command to be executed is not correct.</p>
0x00170011	<p>Command and system status not congruent Command cannot be accepted while the system is in current status.</p>
0x00170012	<p>Wrong parameter command Error displayed when:</p> <ul style="list-style-type: none"> ➢ Error in type of read positions ➢ Wrong origin number during origin pre-setting ➢ Wrong axis number or selection type during axis selection ➢ Error in vedor definition during thickness cutter compensation ➢ Error on command during Multi Block Retrace or Search in Memory
0x00170013	<p>Unattended system reply Switch off the system, then reboot. If problem persists please contact Customer Assistance.</p>

Code (HEX)	Description
0x00170032	<p>Wrong number of axes for G code</p> <p>Error displayed when:</p> <ul style="list-style-type: none"> ➢ Programming at least one axis in G04 ➢ Programming only one axis with canned cycles (from G81 to G89) ➢ No axes or too many axes are programmed for probing cycles for G72-G74 ➢ Both axes are given as the starting point of a protected area
0x00170033	<p>Canned cycle parameters missing</p> <p>Error displays some parameters needed to perform the canned cycle are missing (i.e. K, I, ...)</p>
0x00170034	<p>Missing parameters for G code moving</p> <p>Some parameters needed for G code execution are missing (i.e. G33 ... K).</p>
0x00170035	<p>Missing J and/or K for G83 cycle</p> <p>J and/or K parameter for G83 cycle are missing.</p>
0x00170036	<p>Missing I and/or J for G83 cycle</p> <p>I and/or J parameter for G2/G3 (circle)cycle are missing.</p>
0x00170037	<p>Probing cycle parameters missing</p> <p>Some parameters for probing cycle are missing.</p>
0x00170038	<p>Undefined symbol</p> <p>Tried to access a variable that is not configured.</p>
0x00170039	<p>Expression Overflow</p> <p>Mathematical expression is too long (more than 40 characters)</p>
0x0017003A	<p>Function not allowed</p> <p>Error appears when requested:</p> <ul style="list-style-type: none"> ➢ activation of an M code of type program stop while program already stopped or when the system is in HOLD. ➢ an M paramacro configured as an M prefetch ➢ bypass feed request with non-linear active block modification of Master/Slave parameters, but there are no Master/Slave configured
0x0017003B	<p>Illegal use of slave axis</p> <p>A SDA / UDA / XDA command tried to define as Slave an axis that is already a Slave.</p>
0x0017003C	<p>Operand not allowed in canned cycle</p> <p>Operand not allowed for canned cycles G72 G73 G74.</p>
0x0017003D	<p>K parameter not allowed in G84</p> <p>K parameter is no accepted when performing a G84 using a spindle without transducer</p>
0x0017003E	<p>Wrong programming of the circle (G2/G3)</p> <p>Both centre and radius (R) of the circle to programmed has been defined. Remove R value or the operands (I e J) specifying the centre.</p>
0x0017003F	<p>Illegal number of operands</p> <p>Wrong number of operands for LOA trilateral.</p>
0x00170040	<p>Illegal number of pseudo axis</p> <p>Too many pseudo axes programmed into block: maximum number of pseudo axis is 6.</p>
0x00170041	<p>Illegal number of axes in G33 code</p> <p>More than two axis programmed in G33 G code.</p>

Code (HEX)	Description
0x00170043	G not allowed G not allowed in roughing cycle.
0x00170044	Operand not allowed with G code At least one operand is not compatible with the type of current movement
0x00170045	Block and system status not consistent Programmed block is inconsistent with system status: <ul style="list-style-type: none"> ➢ axis migration (GTA) when canned cycle, cutter compensation, dual axis or master / slave are running ➢ axis migration (GTA) when canned cycle, cutter compensation, dual axis or master / slave are running
0x00170046	G active and program state not consistent G function is not consistent with system status. For instance: <ul style="list-style-type: none"> ➢ with cutter compensation active (G41-G42) G operators canned cycle cannot be programmed ➢ threading cannot be programmed when cutter compensation or canned cycles are active ➢ function related to interpolation plane change (G17-G18-G19) cannot be programmed when cutter compensation is active (G41, G42).
0x00170047	Movement type not consistent Current program status and movement type are not consistent
0x00170048	Programming type not consistent Programming type (absolute, incremental or G79) is not consistent with current programming status
0x00170049	Current programming not consistent Current and previous programming are not
0x0017004A	G and dynamic mode not consistent Requested G function is not consistent with current dynamic mode: <ul style="list-style-type: none"> ➢ functions G72, G73, G74 are not accepted while in continuous mode (G27, G28) ➢ it is not possible to change from G27 to G28 (and vice-versa) when non-linear ramp (RAMP > 1) is active
0x0017004B	G41/G42 and part program status not consistent Cutter compensation is not consistent with current part program status (G41/G42).
0x0017004C	G needs spindle with transducer G33 and macro cycle FIL need a spindle with transducer
0x0017004D	G not consistent with feed rate mode G72, G73, G74 must be executed with G94 active
0x0017004E	Operands and program status not consistent Operand is not consistent with the present part program status. For instance: r, b operands are not allowed during ISO standard status (G40)
0x0017004F	Logic operands and program status not consistent Programed logic operands are not consistent with the program status
0x00170050	Logic operands and dynamic mode not consistent Machine logic operands not consistent with the current dynamic mode. Example: M (motion start/end) doesn't match with G27/G28 Cutter diameter compensation active in T (G41, G42)

Code (HEX)	Description
0x00170051	<p>Logic operands and move type not consistent Machine logic operands not consistent with the current motion Example: G33 + M function of end motion</p>
0x00170052	<p>Operand not allowed in the probing cycle If probing cycle is active, operands are forbidden Example: if G37 is active, I,J,K,R,u,v,w,b,t operands programming is forbidden if G72 is active, I,J,K,R,u,v,w,b,t,r operands programming is forbidden</p>
0x00170053	Third helix axis programming not present.
0x00170054	<p>Expedite function without move No move is associated with the M “expedite” in the block. M “expedite” must be associated to a motion.</p>
0x00170055	<p>Feed not programmed Missing feed for a canned cycle.</p>
0x00170056	<p>Speed not programmed Missing spindle feed for canned cycles.</p>
0x00170057	<p>Read only variable TIM, STE and STA variable are read only. They cannot be written.</p>
0x0017005B	<p>Nesting of IF greater than 32 Exceeded maximum nesting of IF commands.</p>
0x0017005C	<p>ELSE not allowed ELSE instruction programmed without a prior IF instruction</p>
0x0017005D	<p>ENDIF not allowed ENDIF instruction programmed without a prior IF instruction</p>
0x0017005E	<p>Part program block too long Current part program block is longer than 127 characters</p>
0x0017005F	<p>Part program access denied Requested part program cannot be accessed since it is currently used by another user.</p>
0x00170060	<p>Program already selected A paramacro, called in MDI, is trying to call itself</p>
0x00170061	<p>Part Program name too long Requested part program name together with its path is longer than 127 characters.</p>
0x00170062	<p>Logic DRIVE not found Specified logical drive was not configured in ODM or in file browser.</p>
0x00170063	<p>Part Program path not found Path specified for part program does not exist.</p>
0x00170064	<p>File not found Part program / routine / paramacro not found</p>
0x00170065	<p>Illegal file name Filename provided to load table (LOA command) is not correct</p>
0x00170066	<p>File access error System cannot load table</p>
0x00170067	<p>Error during part program file handling Control can't upload a Part Program</p>

Code (HEX)	Description
0x00170068	SPG rejected SPG command failed
0x00170069	Part program release rejected Part Program can't be deactivated
0x0017006A	Part program not selected Error triggers when no part program has been activated and user: <ul style="list-style-type: none"> ➢ tries to modify a block in MDI ➢ pushes Cycle Start button, CNC in AUTO or BLK/BLK
0x0017006B	Tool direction and plane compensation not consistent If the thickness tool compensation is active error is displayed when cutter compensation tool direction and plane are inconsistent.
0x0017006C	Duplicated axes in UDA/SDA command Error is displayed when axis name is duplicated within a dual axes command.
0x0017006D	Axis tracing the safety area isn't X nor Y Part Program tracing the safety area doesn't refer to X and Y axes.
0x0017006E	Trilaterals not allowed in safety area definition Part programs defining a safety area cannot contain trilaterals
0x0017006F	M function not allowed M code type not allowed in continuous.
0x00170096	Illegal argument for TAN TAN operator argument is 90 degrees or an odd multiple, thereof the result would be infinite.
0x00170097	Square root of a negative number SQR operator argument is a negative number
0x00170098	Too many programmed axes Error is displayed if: <ul style="list-style-type: none"> ➢ more than 12 axes are programmed in a block ➢ extra-process axis not allowed in UDA/SDA/XDA
0x00170099	Divided by zero Division by zero programmed (for example X 10/0).
0x0017009A	String too long String length cannot exceed 127 characters.
0x0017009B	Label duplicated This message is displayed when the program is selected or activated. It shows that there are two identical labels in the part program.
0x0017009C	Undefined label The label programmed in a branch instruction (GTO) or in a call for a subroutine (EPP) does not exist
0x0017009D	Label too long This message is displayed when the system activates the Part Program. It indicates that a label having more than 6 characters has been programmed
0x0017009E	Too many sub-programs within the Part Program When activating a Part Program, this error indicates that the number of subroutines called by CLS is higher than the maximum number set in AMP. If possible change the selection value using ODM or use CLD to change the calls.
0x0017009F	Too many labels in the Part Program When activating a Part Program, this error indicates that the its label number is higher than the maximum value set in AMP. If possible, change ODM.

Code (HEX)	Description
0x001700A0	<p>File end (FINITO_FILE) Displays the program end for the following operations:</p> <ul style="list-style-type: none"> ➢ Skip ➢ Modify ➢ String search ➢ Program execution
0x001700A1	<p>LF not found (0x0a) at the file end File end code (LF) not found. Modify the program starting a new line after the last one.</p>
0x001700A2	<p>Program start Program start display for the following operations:</p> <ul style="list-style-type: none"> ➢ Skip ➢ Modify ➢ String search
0x001700A3	<p>Nesting of RPT greater than 5 RPT max. nesting level (5) exceeded</p>
0x001700A4	<p>Nesting of subroutine greater than 4 Subroutine max. nesting level (4) exceeded</p>
0x001700A5	<p>Nesting of EPP greater than 5 EPP max. nesting level (5) overflow</p>
0x001700A6	<p>RPT/ERP cycle open at end of file This message is displayed when:</p> <ul style="list-style-type: none"> ➢ The end of the file has been reached without finding the (ERP) block that closes the programmed (RPT) cycle ➢ The end of the file has been reached without completing the subroutine defined with (ERP)
0x001700A7	<p>ERP without RPT (ERP) has been programmed without previously programming (RPT)</p>
0x001700A8	<p>Modal Paramacro already active A modal G paramacro is programmed when a modal G paramacro is already active.</p>
0x001700A9	<p>Paramacro not configured A missing G paramacro is programmed.</p>
0x001700AD	<p>Too many elements in safety area definition Safety areas should be defined using no more than 10 (closed areas) or 9 (open areas) geometric elements.</p>
0x001700AE	<p>Safety area could not be closed An open safety area has been defined using 10 geometric elements, CNC cannot close it.</p>
0x001700AF	<p>Active safety area cannot be redefined User is trying to redefine a safety area. Safety areas can be redefined only if not active.</p>
0x001700B0	<p>Error in limits for 3D safety area Higher and lower limits for 3D safety area do not belong to the same axis.</p>
0x001700B1	<p>XMLrules.txt not loaded. XML commands not executed XML paramacros cannot be executed since no mapping file has been loaded. Provide a valid mapping file (XMLrules.txt).</p>

Code (HEX)	Description
0x001700B2	Syntax error in XMLrules.txt Syntax error in XML-paramacro correspondences file
0x001700FA	<p>Axis not referenced This message occurs when:</p> <ul style="list-style-type: none"> ➢ The programmed axis is not referenced ➢ The offset to be pre-set/re-qualified is associated with a non-referenced axis.
0x001700FB	<p>Undefined DPP for probing cycle Probing cycle parameters (approach coordinate, safety distance, velocity) are not defined in the DPP block</p>
0x001700FC	<p>Too many "Expedite" M codes More than one expedite M code has been programmed in the block</p>
0x001700FD	<p>Undefined M code Programmed M code is not configured in AMP. Configure the M in AMP using ODM tool.</p>
0x001700FE	<p>Circle not consistent The circle is not geometrically consistent: radius or final point are not correct</p>
0x001700FF	<p>Wrong threading parameters Threading parameters not consistent. Calculate the I parameter with the following formula:</p> $\frac{16k}{2(\text{threading distance})}$
0x00170100	<p>Helix pitch incongruent The helix pitch is not geometrically correct</p>
0x00170101	<p>Different SCF in the interpolation plane axes. In the G02/G03 (circle) programming, axes must have the same scale factor. Change scale factor using SCF.</p>
0x00170102	<p>Not consistent geometry Wrong programming of an ISO profile</p>
0x00170103	<p>Inconsistent compensated profile After compensation, programmed profile is not continuous anymore.</p>
0x00170104	<p>Wrong profile direction Cutter diameter compensation value (G41/G42) changes the direction on the profile.</p>
0x00170105	<p>Compensation closing error Cutter diameter compensation (G40) wrong exit.</p>
0x00170106	<p>Full transit buffer Error occurs if: when programming with tool compensation active, (G41 - G42) too many motions outside the interpolation plane have been programmed interferences offset is enabled (IBSIZE not 0) too many interferences generators are rejected</p>
0x00170107	<p>Too many blocks to release Monitor displays the full input buffer message. Please, contact the Customer Assistance.</p>
0x00170108	<p>Safety area entry The programmed move enters one of the active safety areas</p>

Code (HEX)	Description
0x00170109	Canned cycle on rotated plane Canned cycle programmed on rotated plane Disable plane rotation
0x0017010A	Canned cycle length data incongruent The parameters for whole depth, approach dimension, return after machining is not consistent during a canned cycle.
0x0017010B	Canned cycle data incongruent The parameters specified in the canned cycle (I, J, K, R) are not allowed. For example: canned cycle K = 0
0x0017010C	Unexpected canned cycle
0x0017010D	Wrong probing cycle programming This message appears when: probing approach distance is null hole probing is programmed with null radius (for example G73r0E5)
0x00170110	Axes not on profile This message appears when trying to quit CYCLE STOP (HOLD) after a series of manual moves without taking the axes back to the profile. Select JOG RETURN and return the axes to the profile
0x00170111	Error in exit HOLD: mode changed This message occurs when trying to exit from HOLD by setting an operating mode that is different from the one in which the system went on HOLD. Select the right mode and retry.
0x00170112	Block not allowed in HOLD This message occurs when trying to execute an MDI block that is not allowed in HOLD
0x00170114	Wrong use of roll-over axis with G90 The programmed coordinate for the axis with rollover in G90 is greater than the roll over pitch configured in AMP
0x00170117	Command not allowed in RCM
0x00170118	Output elements buffer is full
0x00170119	Axis shared with logic Axis shared with PLC cannot be moved in jog mode or returned on profile
0x0017011A	Safety area is not convex Safety areas must be convex.
0x0017011B	Undefined safety area. Activation impossible Request to activate a safety area not previously traced by DPA.
0x0017011C	Undefined safety area. Activation impossible Trying to translate a safety area on an axis that does not belong to the area.
0x0017011D	Axis type not consistent for axis tangential control At least one of the axis needed by axis tangential control is not correct. Two linear axis and one rotary axis are needed.
0x0017011E	Automatic return to profile not allowed Automatic return to profile is not allowed after the execution of an MDI command in HOLD state

Code (HEX)	Description
0x0017015E	<p>Mode to select out of range This message occurs when the selected mode is out of range. Allowed modes are in the 1-8 range: MDI AUTO BLOCK by BLOCK CONTINUOUS JOG INCREMENTAL JOG RETURN ON PROFILE HOMING FILE HPG (Hand Wheel)</p>
0x0017015F	<p>Axes number to select out of range The error triggers in two cases: The number of axes selected with MIR function is higher than the number of axes available to the process The number of axes selected for manual moves with library call is out of range. The allowed range is from 1 to the number of axes configured for the process</p>
0x00170160	<p>Too many axes selected for manual move A larger number of axis names than accepted have been inserted in the part program block. Edit the part program block.</p>
0x00170161	<p>Mode selected and cycle not consistent This error is displayed when CYCLE START is pressed in the following conditions: <ul style="list-style-type: none"> ➢ system in HRUN with MBR active and selected mode other than AUTO or BLK/BLK ➢ a mode other than MDI has been selected during execution of system axes moves ➢ system in IDLE with MBR active and selected mode other than AUTO or BLK/BLK ➢ system on HOLD, AUTO or BLK/BLK with MBR not configured in AMP ➢ system on HOLD with MBR active and selected mode other than AUTO or BLK/BLK NOTE: For further information about the machine refer to the user guide </p>
0x00170162	<p>Data length out of range Error message is displayed if: MDI block length is out of range Error on library call for axis sharing activation</p>
0x00170163	<p>Operative limit definition wrong Error in defining the software operating limits with the three-letter mnemonic SOL.</p>
0x00170164	<p>Offset length not defined for the axis This message occurs when trying to pre-set or to re-qualify an offset that is not associated with the axis specified.</p>
0x00170165	<p>Tool orientation code wrong The specified tool orientation code is wrong.</p>

Code (HEX)	Description
0x00170166	Error returned from Axis call Internal error. If monitor displays the message, please contact the Customer Assistance.
0x00170167	Manual motion failed: no axis configured Homing and return on profile cannot be performed as no axis has been selected.
0x00170168	Axis not configured for per GTA/GTS GTA or GTS Id is not configured or in GTS a non-spindle axis has been programmed.
0x0017016A	Axis not available Axis to share with the logic is unavailable.
0x0017016B	Duplicated axis ID Axis ID is duplicated in the GTA programming.
0x0017016C	Axis Id incompatible As GTA axis ID refers to a spindle, it can't be used.
0x0017016D	Too many tools involved More than 64 tools have been set in the programming
0x00170190	Activation/variation of the spindle rotation failed Machine doesn't accept spindle feed rate change.
0x00170191	New tool request failed Machine doesn't accept the T code programming.
0x00170192	M execution failed Machine doesn't accept the M programming or code execution failed.
0x00170193	Pseudo axes programming failed Machine logic does not accept the programmed pseudo axis.
0x00170194	Motion request denied Logic machine doesn't allow any motion.
0x00170195	End motion failed Logic machine doesn't accept the end motion message.
0x00170196	Too many blocks programmed without motion in continuous mode start
0x001701C2	Axis on profile Displays axis is on the profile and return on profile operation was successful.
0x001701C3	End of automatic return on profile The return on profile operation is ended correctly and all the axes are back on the profile.
0x001701C4	Block retrace end The backward path execution (MBR) is finished. To execute more block the change of configuration is needed.
0x001701C7	Search memory ended
0x001701F4	Current process number programmed Process number for synchronisation instructions identifies the current process.
0x001701F5	ASSET option disabled ASSET is not implemented in the control.
0x001701F6	Undefined geometric element
0x001701F7	Spindle shared. Impossible canned cycle. G84 or G86 programmed while spindle isn't in exclusive mode.

Code (HEX)	Description
0x001701F8	Spindle required for a canned cycle Programmed canned cycle needs a spindle, but no spindle is available for the current process.
0x001701F9	Dry Run active Dry Run requested when already active.
0x001701FA	Process non-existent A multi-process command tries to communicate with a process that is not defined
0x001701FC	Queue is full on target process The process queue receiving the message is full.
0x001701FD	Data sending too long Data transmitted using SND exceed 576 characters.
0x001701FE	Data loading failed The type or number of data transmitted with SND is not consistent with the forecasted one.
0x001701FF	Message still in queue A SND command has been given before the process cleared the previous message.
0x00170200	EXE or ECM failed The status of the process to which the EXE or ECM command is sent does not allow automatic part program execution commands (RUN, HRUN, RUNH, HOLD) or an MDI instruction. There is a syntax error in the program to which the EXE command is addressed
0x00170233	Axis activation/release error GTA is not allowed when Offset and Canned cycles are active.
0x001A0001	Circles/lines not intersecting No intersection found for the circle / lines. Check that initial and final point and radius or centre of the circular movement are correct.
0x001A0005	Parallel lines The requested connection between two elements cannot be established. Connection can be programmed or generated by path optimization during cutter compensation (TPO = 33).
0x001A0006	Wrong elements definition Internal error, if displayed on monitor, please contact the customer assistance.
0x001A0007	Aligned points The points measured during the measure of the parameters of an hole (G73) do not trace a circle.
0x001A000A	Different radius between initial-final points Radius of circular movements on starting point differs from the one computed on final point. Check centre, starting and ending point of the circular movement.
0x001A000B	Circle incorrect Normalisation of circular movement failed. Check starting, ending points and centre or radius of the circular movement. Verify tolerance for variance of circle points (CET) and arc normalisation mode (ARM).

3. OPEN_MOTION LIBRARY: motion functions

GMC_InterpCreate	Creates an interpolator for use by the PLC
GMC_InterpDelete	Removes a PLC interpolator
GMC_InterpEnterFeedHold	Sets an interpolator to feedhold status
GMC_InterpExitFeedHold	Ends the feedhold status of an interpolator
GMC_InterpEnterHold	Requires an interpolator to enter the “hold” status
GMC_InterpExitHold	Ends the “hold” status of an interpolator.
GMC_EnterFeedHold	Sets all PLC interpolators to feedhold
GMC_ExitFeedHold	Ends the feedhold state for all PLC interpolators
GMC_EnterHold	Sets all PLC interpolators to hold
GMC_ExitHold	Ends the “hold” status for all the PLC interpolators.
GMC_InterpModify	Changes the configuration of the interpolator
GMC_InterpReadInfo	Reads the status of an interpolator
GMC_InterpReset	Asks a PLC interpolator to reset itself
GMC_Reset	Asks all PLC interpolators to reset themselves
GMC_Move	Executes a linear movement
GMC_MoveJog	Executes the manual movement in jog
GMC_MoveOnPosition	Executes a linear movement conditional on the position of the axis
GMC_MoveOnSignal	Executes a linear movement conditional on the signal status
GMC_MoveUntilSignal	Executes linear movements ending on signal
GMC_Poly	Executes a polynomial interpolation
GMC_ProtectAreaCmd	Works on a protected area previously defined
GMC_MoveRoundOFF	Executes a linear movement with “release”
GMC_MoveUntilVariable	Executes a linear movement that ends on the basis of the value of a variable
GMC_ToleranceParams	Defines the tolerance for the calculation of circular trajectories
GMC_CircleCCW	Executes a counter clockwise circular movement with a given centre
GMC_CircleCCWRadius	Executes a counter clockwise circular movement With a given centre
GMC_CircleCW	Executes a clockwise circular movement with a given centre
GMC_CreateAxVirtual	Creates a virtual axis among the axes managed by the interpolator
GMC_DeleteAxVirtual	Deletes the virtual axis from the table of the axes managed by the interpolator
GMC_CircleCWRadius	Executes a clockwise circular movement with a given centre
GMC_ContinuousAbort	Determines the ends of a continuous motion and its cancellation
GMC_ContinuousEnd	Determines the closure of a continuous stage and its cancellation
GMC_ContinuousParallel	Determines the closure of a continuous stage
GMC_ContinuousParam	Activates continuous parallel motions
GMC_ContinuousStart	Activated the continuous motion
GMC_HwndExe	Executes axes movement through hand wheel

GMC_HwndOFF	Disables axes movement through hand wheel
GMC_HwndON	Enables axes movement through hand wheel
GMC_HwndParam	Defines application parameters of the hand wheel on the axes
GMC_HwndParamExt	Defines the handwheel application parameters on the axes
GMC_MSLdefine	Executes the master/slave association
GMC_MSLexe	Immediately activated the master/slave following
GMC_MSLexeONposition	Activates mater/slave following on axis position
GMC_MSLexeONsignal	Activates the mater/slave following on signal
GMC_MSLfollow	Immediately modifies master/slave following ratio
GMC_MSLfollowONposition	Modifies the master/slave following ratio on axis position
GMC_MSLfollowONsignal	Modifies the master/slave following ration on signal
GMC_MSLjog	Immediately activates JOGGING for a slave
GMC_MSLjogONposition	Activates JOGGING for a slave on axis position
GMC_MSLjogONsignal	Activates JOGGING for a slave on axis
GMC_MSLparam	Defines the parameters of master/slave axes following
GMC_MSLreset	Immediately reset the value of the master compared to the slave
GMC_MSLresetONposition	Immediately reset the value of the master compared to the slave based on the axis position
GMC_MSLresetONsignal	Resets the master value compared to a slave based on a signal
GMC_MSLundef	Executes the dissociation between master/slave axes
GMC_MSLrealign	Realigns the position of a Slave with reference to the Master
GMC_Probe	Executes probing cycle
GMC_ProtectAreaDefine	Defines a protected area
GMC_ReadAxisInfo	Reads the status of an axis
GMC_ReadAxisInfoALL	Reads the status of all axes
GMC_ChangePosLoop	Modifies the servo axis mode
GMC_ChangePosLoopALL	Modifies the servo mode for all axes
GMC_SetAxisProperty	Activates the axes set up mode
GMC_SetAxisPropertyALL	Activates the axes set-up mode for all the axes
GMC_SetBitONposition	Sets a signal condition by axis position
GMC_SetBitONvariable	Sets a signal condition by value of a variable
GMC_SetFeed	Defines feed rate on the profile
GMC_SetFeedOverride	Defines feed rate on the profile
GMC_SetFeedOverrideALL	Determines the feed % for the axes specified
GMC_SetJerk	Defines the S ramps JRK
GMC_SetRamp	Changes the ramp mode for the axes
GMC_SetRampALL	Modifies the ramp mode for all the axes
GMC_SetOrigin	Configures an origin on the axes
GMC_ActivateOrigin	Activates an origin
GMC_SetResetQuote	Immediately configures the axis reset value
GMC_ActivateResetQuote	Resets an axis
GMC_SetResetQuoteONmarker	Configures axis reset on respect to the marker
GMC_SetResetQuoteONPosition	Configures the axis reset point as a function of axis Position
GMC_SetResetQuoteONsignal	Configures the axis reset point influenced by a signal
GMC_SetResetQuoteShift	Configures the value of reset axis position change
GMC_Homing	Executes the axis reset

GMC_SpindleChangeGear	Performs the gear change on the spindle
GMC_SpindleG96	Starts the spindle rotation at a certain constant cutting speed
GMC_SpindleG97	Starts the spindle rotation at a certain speed in rpm
GMC_SpindleOrient	Waits on axis position
GMC_WaitONposition	Waits for value of a signal
GMC_WaitONsignal	Determines a certain waiting period
GMC_WaitONtimeout	Waits for a value of a variable
GMC_WaitONvariable	Waits on axis position
GMC_Drill	Executes drilling cycle
GMC_Bore	Executes boring cycle
GMC_Tapp	Executes tapping cycle without transductor
GMC_TappTransd	Executes tapping cycle with transductor
GMC_Thread	Executes threading cycle
GMC_Circle3DCCW	Executes an anticlockwise circular motion in the space known the centre and the arriving point
GMC_Circle3D3P	Executes a circular motion for three given points
GMC_Circle3DCW	Executes a clockwise circular motion in the space known the centre and the arriving point
GMC_SetDynamicOverride	Selects dynamic override percentage for specified axes
GMC_SetDynamicOverrideALL	Selects dynamic override percentage for all axes associated to the PLC
GMC_CamAxiDefine	Defines an electronic cam associated to a master and to a slave axis
GMC_CamAxiStart	Enables one or more electronic cams mater/slave type
GMC_CamAxiStartOnPosition	Enables one or more electronic cams slave/master type, according to the value of the AxisId axis position
GMC_CamAxiStartOnSignal	Enables one or more electronic cams master/slave type on signal
GMC_CamOutDefine	Defines an electronic cam associated to a master to to the outputs.
GMC_CamOutStart	Enables an electronic cam output type
GMC_CamOutStartOnPosition	Enables an electronic cam output type
GMC_CamOutStartOnSignal	Enables one or more lectronic camsmaster/slave type on signal
GMC_CamStop	Disables one or more active cams
GMC_CamStopOnSignal	Disables one or more cams active on signal
GMC_CamStopOnPosition	Disables one or more active cams according tot an axis position
GMC_CamUndefine	Deletes one or more input cams

3.1 GMC_InterpCreate

Function **GMC_InterpCreate** creates an interpolator for use by the PLC.

Syntax:

```
ret_code := GMC_InterpCreate (InterpId, Index, ErrMode, EmgMode, PntMode) ;
```

Input parameters:

- ErrMode* (WORD) Error management mode
EmgMode (WORD) Emergencies management mode
PntMode (WORD) Coordinate (point) management mode

Output parameters:

- InterpId* (INT) Interpolator identifier PLC [1..40]
Index (INT) Interpolator index PLC [1..40]

Execution mode:

Immediate

Use:

This function makes it possible to create a PLC interpolator. With this interpolator PLC can execute the movements of one or more axes, synchronism waits, etc. The outputs include the identifier of the newly created interpolator (*InterpId*) and a number defining the position of the descriptive data of the interpolator within variables PW and PD (*Index*). The *InterpId* must be used in all movement calls so as to assign the movement to a well-defined interpolator. Variable *Index* must be used to access the interpolator data as follows (see application manual):

(*Index* * 40) + 4000 Index of interpolator status area start variable PW
 (*Index* * 40) + 4000 Index of interpolator dynamic data area start variable PD

Using call **GMC_InterpReadInfo** it is possible to access in real time all interpolator status and data without having to use the direct access to variables PW and PD.

The behaviour of the interpolator is strongly characterised by the following configuration parameters:

ErrMode determines system operating modalities in the event of an error generated by an interpolator

ErrMode	Mnemonic	Description
0	ERR_NORMAL	All movements underway are ended and interpolator status changes to ERROR. It is necessary to request an interpolator Reset to be able to reuse the same interpolator in subsequent operations.
1	ERR_CALLER	The interpolator carries on its activities and it is up to the caller (PLC) to stop the interpolator operations underway - if any - through a Reset call.

2	ERR_EMERG	All movements underway are ended and interpolator status changes to EMERGENCY. The emergency is propagated to the entire system and may be captured by the ad hoc filter task. It is necessary to request an interpolator Reset to be able to reuse the same interpolator in subsequent operations.
---	-----------	---

EmgMode defines the operating modalities of an interpolator in the event of an emergency being sent to it (emergency that generally involves axes under its control or devices for general use by the CNC).

EmgMode	Mnemonic	Description
0	EMG_NORMAL	All movements underway are ended and interpolator status changes to EMERGENCY. It is necessary to request an interpolator Reset to be able to reuse the same interpolator in subsequent operations.

PntMode defines whether the axis points programmed on the axes associated with the interpolator must take into account possible offsets/origins present on the axes.

EmgMode	Mnemonic	Description
0	POS_WITHOFS	The offset values and/or the origins associated with the axes are added to the axis positions.
1	POS_NOOFS	The axis points programmed must be construed as absolute, and therefore not affected by origins and/or offset values.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160011	Interpolator type wrong (internal error)
0x00160015	Interpolator already present (internal error)
0x00160012	Error management mode wrong
0x00160013	Emergency management mode wrong
0x00160010	Point management mode wrong
0x00160001	Too many interpolators created

Example:

```
Ret      : dword;
InterpId : INT ;
Index    : INT ;
Ret :=GMC_InterpCreate (InterpId, Index, ERR_NORMAL, EMG_NORMAL, POS_NOOFS) ;
```

See also:

[GMC_InterpDelete](#), [GMC_InterpModify](#), [GMC_InterpReset](#), [GMC_InterpReadInfo](#).

3.2 GMC_InterpDelete

Function GMC_InterpDelete removes a PLC interpolator.

Syntax:

```
ret_code := GMC_InterpDelete (InterpId) ;
```

Input parameters:

InterpId (INT) Interpolator identifier PLC [1..40]

Execution mode:

Immediate

Use:

This function is used to destroy the interpolator whose identifier is *InterpId*. The interpolator must not have any activity pending (use function GMC_InterpReadInfo to verify IDLE or RESET status), otherwise an error is returned.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160016	Interpolator with activities pending
0x00160017	Interpolator with movements pending

Example

```
Ret      : dword;
InterpId : INT;
...
Ret := GMC_InterpDelete (InterpId);
```

See also

GMC_InterpCreate, GMC_InterpModify, GMC_InterpReset, GMC_InterpReadInfo.

3.3 GMC_InterpEnterFeedHold

Function **GMC_InterpEnterFeedHold** sets an interpolator to FeedHold status.

Syntax:

```
ret_code := GMC_InterpEnterFeedHold (ExecMode, ExecStatus, InterpId, FhldMode) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

InterpId (INT) Interpolator identifier PLC [1..40]

FhldMode (INT) Input mode in FeedHold

Execution mode

Wait / NoWait

Use:

The interpolator specified is set to FeedHold status, i.e., all movements underway (and subsequent movements) are set to 0 feed. If no movement is active, the interpolator enters FeedHold status immediately. At any rate, the movement remains active and will be resumed upon receiving an exit FeedHold request. The FeedHold entry mode depends on parameter *FhldMode*, which may be:

FhldMode	Mnemonic	Description
1	FHOLD_NORMAL	The movements underway (or subsequent movements) are requested to go to 0 feed rate immediately.
2	FHOLD_ENMOV	The movement underway is requested to go to nil speed at end of movement. If a movement cannot go to 0 feed at the end of the movement underway, it will do so as soon as possible immediately after the end of the current movement (same behaviour as with command FHOLD_NORMAL).



The command is rated as over the moment the interpolator acquires the request, not when the movement reaches nil feed rate.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160011	Interpolator type wrong (internal error)
0x00160014	Interpolator not present
0x0016001D	FeedHold performance mode wrong

Example:

```
Ret      : dword;
InterpId : INT ;
...
Ret := GMC_InterpEnterFeedHold (MODE_WAIT, UL0, InterpId, FHLD_NORMAL);
```

See also:

[GMC_InterpExitFeedHold](#), [GMC_EnterFeedHold](#), [GMC_ExitFeedHold](#).

3.4 GMC_InterpExitFeedHold

Function GMC_InterpExitFeedHold ends the FeedHold status of an interpolator.

Syntax:

```
ret_code := GMC_InterpExitFeedHold (ExecMode, ExecStatus, InterpId) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

InterpId (INT) Interpolator identifier PLC [1..40]

Execution mode:

Wait / NoWait

Use:

This request ends the FeedHold status for the interpolator specified. The execution of the movements present, if any, is resumed. The command is accepted even if the interpolator is not in FeedHold status; in this case however it has no effect on the movements underway.



This command is deemed to be over as soon as the interpolator acquires the request, not when the movement is resumed.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160011	Interpolator type wrong (internal error)
0x00160014	Interpolator not present

Examples:

```
Ret      : dword;
InterpId : INT ;
...
Ret := GMC_InterpExitFeedHold (MODE_WAIT, ULO, InterpId);
```

See also:

GMC_InterpEnterFeedHold, GMC_EnterFeedHold, GMC_ExitFeedHold.

3.5 GMC_InterpEnterHold

Function GMC_InterpEnterHold requires an interpolator to enter the “hold” status.

Syntax:

```
ret_code := GMC_InterpEnterHold (ExecMode, ExecStatus, InterpId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]

Execution mode:

Wait / NoWait

Use:

The interpolator is set to Hold mode and all motions are set to speed 0; if there are no active movements, the entry into Hold takes place immediately. The interpolator continues to be active and will be resumed upon a request to exit Hold.



This command is deemed to have ended the moment the interpolators have acquired it, not when the movements reach zero speed. The status of the interpolators must be checked by means of function GMC_InterpreadInfo.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160011	Wrong interpolator (internal error)
0x00160014	Missing interpolator
0x00160019	Instruction cannot be executed due to: interpolator in error or emergency status or performing a reset.

Example:

```
Ret      : dword;
InterpId : INT ;
...
Ret := GMC_InterpEnterHold (MODE_WAIT, ULO, InterpId);
```

See also:

GMC_InterpExitHold, GMC_EnterHold, GMC_ExitHold, GMC_InterpReset, GMC_Reset

3.6 GMC_InterpExitHold

GMC_InterpExitHold function releases an interpolator Hold status.

Syntax:

```
ret_code := GMC_InterpExitHold (ExecMode, ExecStatus, InterpId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution mode
<i>InterpId</i> (INT)	Identifier interpolator PLC [1..40]

Execution mode:

Wait / NoWait

Use:

This instruction releases the hold status of a specified interpolator. All the movement restart immediately their execution.



The command is regarded as concluded the moment the interpolators have acquired the request, not when the movements are resumed.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160011	Wrong interpolator (internal error)
0x00160014	Interpolator missing
0x0016001A	Command not executed: the interpolator is not in HOLD status

Example:

```
Ret      : dword;
InterpId : INT ;
...
Ret := GMC_InterpExitHold (MODE_WAIT, ULO, InterpId);
```

See also:

GMC_InterpEnterHold, GMC_EnterHold, GMC_ExitHold. Gmc_InterpReset, GMC_Reset.

3.7 GMC_EnterFeedHold

Function `GMC_EnterFeedHold` sets all PLC interpolators to FeedHold

Syntax:

```
ret_code := GMC_EnterFeedHold (ExecMode, ExecStatus, FhldMode) ;
```

Input parameters:

`ExecMode` (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
`ExecStatus` (DWORD) Command execution status
`FhldMode` (INT) Enter mode in FeedHold

Execution mode

Wait / NoWait

Use:

All PLC interpolators are set to FeedHold, i.e., all current (or subsequent) movements are set to speed 0. At any rate, the movements continue to be active and will be resumed upon a request to exit FeedHold. The entry into FeedHold mode depends on parameter `FhldMode`, which may be:

FhldMode	Mnemonic	Description
1	FHOLD_NORMAL	Current (or subsequent) movements are requested to reduce their speed to zero.
2	FHOLD_ENMOV	The movements underway are requested to reduce their speed to zero at end of move. If zero speed cannot be obtained during the move underway, it will be obtained as soon as possible after the end of the current move (same behaviour as with command FHOLD_NORMAL).



This command is deemed to have ended the moment the interpolators have acquired it, not when the movements reach zero speed.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without error
0x0016001D	Execution mode of wrong FeedHold

Example:

```
Ret : dword;
Ret := GMC_EnterFeedHold (MODE_WAIT, ULO, FHLD_NORMAL);
```

See also:

`GMC_InterpExitFeedHold`, `GMC_InterpEnterFeedHold`, `GMC_ExitFeedHold`.

3.8 GMC_ExitFeedHold

Function GMC_ExitFeedHold ends the FeedHold state for all PLC interpolators.

Syntax:

```
ret_code := ExitFeedHold (ExecMode, ExecStatus) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT,
MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Execution status:

Wait / NoWait

Use:

This request concludes the FeedHold condition for all PLC interpolators. The execution of any movement present is resumed. This command is accepted even if an interpolator is not in FeedHold state: in this case it has no effect on the movements underway.



The command is regarded as concluded the moment the interpolators have acquired the request, not when the movements are resumed.

Return values

The following table gives the possible values values of variable *ret_code*:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;  
Ret := GMC_ExitFeedHold (MODE_WAIT, UL0);
```

See also:

GMC_InterpEnterFeedHold, GMC_EnterFeedHold, GMC_InterpExitFeedHold.

3.9 GMC_EnterHold

Function GMC_EnterHold sets all PLC interpolators to Hold.

Syntax:

```
ret_code := GMC_EnterHold (ExecMode, ExecStatus) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT,
MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Execution mode:

Wait / NoWait

Use:

All PLC interpolators are set to Hold, i.e., all current (or subsequent) movements are set to speed 0; if there are no active movements, entry into Hold takes place immediately (i.e., new interpolators can take over the axes moved by the interpolators in Hold) the movements continue to be active and will be resumed upon a request to exit Hold.



This command is deemed to have ended the moment the interpolators have acquired it, not when the movements reach zero speed. The status of the interpolators must be checked by means of function GMC_InterpreadInfo.

Return values

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160019	Command cannot be executed due to: interpolator in error or emergency status or performing a reset.

Example:

```
Ret : dword;
Ret := GMC_EnterHold (MODE_WAIT, UL0);
```

See also:

GMC_InterpExitHold, GMC_InterpEnterHold, GMC_ExitHold, GMC_InterpReset, GMC_Reset.

3.10 GMC_ExitHold

Function GMC_ExitHold ends the Hold status for all the PLC interpolators.

Syntax:

```
ret_code := GMC_ExitHold (ExecMode, ExecStatus) ;
```

Input parameter:

<i>ExecMode</i> (INT)	Execution mode command [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Execution status command

Execution mode:

Wait / NoWait

Use:

This instruction releases all the PLC interpolators from the HOLD status. All the movements restart immediately.



The command is regarded as concluded the moment the interpolators have acquired the request, not when the movements are resumed.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0016001A	Command not executed: the interpolator is not in HOLD status

Example:

```
Ret : dword;
Ret := GMC_ExitHold (MODE_WAIT, UL0);
```

See also:

GMC_InterpEnterHold, GMC_EnterHold, GMC_InterpExitHold, Gmc_InterpReset, GMC_Reset.

3.11 GMC_InterpModify

Function **GMC_InterpModify** changes the configuration of the interpolator.

Syntax:

```
ret_code := GMC_InterpModify (ExecMode, Interpld, ErrMode, EmgMode, PntMode) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>ErrMode</i> (WORD)	Error management mode
<i>EmgMode</i> (WORD)	Emergencies management mode
<i>PntMode</i> (WORD)	Co-ordinates (points) management mode

Execution mode:

Wait / NoWait

Use:

This function makes it possible to modify the operating characteristics of a previously created interpolator. The following configuration parameter may be modified: *ErrMode* determines system behaviour in the event of an error generated by an interpolator.

ErrMode	Mnemonic	Description
0	ERR_NORMAL	All movements underway are ended and interpolator status changes to ERROR. It is necessary to request an interpolator Reset to be able to reuse the same interpolator in subsequent operations.
1	ERR_CALLER	The interpolator carries on its activities and it is up to the caller (PLC) to stop the interpolator operations underway, if any, through a Reset call.
2	ERR_EMERG	All movements underway are ended and interpolator status changes to EMERGENCY. The emergency is propagated to the entire system and may be captured by the ad hoc filter task. It is necessary to request an interpolator Reset to be able to reuse the same interpolator in subsequent operations.

EmgMode defines the operating modalities of an interpolator in the event of an emergency being sent to it (emergency that generally involves axes under its control or devices for general use by the CNC).

EmgMode	Mnemonic	Description
0	EMG_NORMAL	All movements underway are ended and interpolator status changes to EMERGENCY. It is necessary to request an interpolator Reset to be able to reuse the same interpolator in subsequent operations.

PntMode defines whether the axis points programmed on the axes associated with the interpolator must take into account possible offsets/origins present on the axes.

PntMode	Mnemonic	Description
0	POS_WITHOFS	The offset values and/or the origins associated with the axes are added to the axis positions.
1	POS_NOOFS	The axis points programmed must be construed as absolute, and therefore not affected by origins and/or offset values.

Return values:

The following table gives the possible values of variable *ret_code*:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160011	Interpolator type wrong (internal error)
0x00160014	Interpolator not present
0x00160012	Error management mode wrong
0x00160013	Emergencies management mode wrong
0x00160010	Dimensions management mode wrong

Example:

```
Ret      : dword;
InterpId : INT ;
...
Ret := GMC_InterpModify (MODE_WAIT, UL0, InterpId, ERR_NORMAL,
                        EMG_NORMAL, POS_NOOFS);
```

See also:

[GMC_InterpDelete](#), [GMC_InterpCreate](#), [GMC_InterpReset](#), [GMC_InterpReadInfo](#).

3.12 GMC_InterpReadInfo

Function GMC_InterpReadInfo reads the status of an interpolator.

Syntax:

```
ret_code := GMC_InterpReadInfo (InterpId, Info) ;
```

Input parameters:

InterpId (INT) Interpolator identifier PLC [1..40]

Output parameters:

Info (InterpInfo_Struct) Structure in which interpolator data is read

Execution mode:

Immediate

Use:

The data associated with an interpolator PLC are loaded into a type InterpInfo_Struct structure. If the interpolator requested is not present, the structure is loaded with data set to 0; if the TaskID field within the structure is set to 0, the interpolator requested is non-existent. The data contained in this structure are:

Field	Type	Description
TaskSts	WORD	Executive status of interpolator
TaskSub	WORD	Executive sub-state (for phase of INT_ST_STOP)
TaskPrgSts	WORD	Interpolator programming mode
TaskID	WORD	Id of task with which interpolator is associated
TaskErr	DWORD	Error present on interpolator
ConfSts	WORD	Configuration flag
ContSts	WORD	Continuous mode status
StoredBlock	DWORD	Number of movements in continuous mode analysed
ExecudBlock	DWORD	Number of movements in continuous mode executed
Odh	WORD	Execution status (debug)
RunSts	WORD	Word Status
FholdSts	WORD	FeedHold Status
ActVel	LREAL	Current feed rate (unit/min)
ActVell	LREAL	Current feed rate (unit/clock)
DistToGo	LREAL	Distance To Go to end of block being executed
DistCov	LREAL	Distance covered in block being executed
Angle	LREAL	Current angle for circular motion (in radians)
TimeToGo	LREAL	Time To Go to end of block in execution (in ms) Valid only for NON continuous motions
DistToGoHold	LREAL	Distance To Go to end of block in hold
DistCovHold	LREAL	Distance covered in block in hold
AngleHold	LREAL	Current angle for circular motion (in radian) in hold

This information will be available starting from the OPENcontrol release 2.1:

Field	Type	Description
DistCovGlb	LREAL	Distance covered for the begin of the profile

Starting from the OPENcontrol release 3.2, the following information will be also available

Field	Type	Description
MovStat	WORD	Movement status
MovSubs	WORD	Movement substatus

For some of the fields described:

TaskSts	Mnemonic	Description
0	INT_ST_FREE	Interpolator not existent
1	INT_ST_IDLE	Interpolator not active
2	INT_ST_RUN	Interpolation in progress
3	INT_ST_STOP	Interpolator in stop mode
4	INT_ST_HOLD	Interpolator in Hold
5	INT_ST_ERR	Interpolator in error
6	INT_ST_EMG	Interpolator in emergency
7	INT_ST_RESE	Interpolator in reset

TaskSub	Mnemonic	Description
0	INT_ST_STOH	Interpolator braking for Hold
1	INT_ST_STOR	Interpolator braking for Reset
2	INT_ST_STOE	Interpolator braking for Emergency
3	INT_ST_STOX	Interpolator braking for Error
4	INT_ST_STOQ	Interpolator braking for Stop request

TaskPrgSts	Mnemonic	Description
0	INT_PRG_IDLE	No active command
1	INT_PRG_CONT	Continuous programming
2	INT_PRG_ABOC	Continuous programming aborted
3	INT_PRG_RUNN	Wait for command in execution to end
4	INT_PRG_QUEU	Continuous programming with full queue
5	INT_PRG_SPIND	Programming of operations on spindle

ConfSts	Mnemonic	Description
0x0010	INT_ERR_MANAG	Errors managed by PLC
0x0020	INT_ERR_STOP	Interpolation stop on programming error
0x0040	INT_ERR_EMG	Generates an emergency for programming error
0x0100	INT_PRG_NOOFS	Programmed points are absolute
0x4000	INT_PLC	PLC interpolator
0x8000	INT_CNC	CNC interpolator

ContSts	Mnemonic	Description
0x0001	(CNT_RUN_OK):	Continuous run underway
0x0002	(CNT_RUN_DEF):	Continuous run in lack of entities status
0x0004	(CNT_ABORT):	Continuous in abort status

Odh	Mnemonic	Description
0x0001	(ODH_DEFG):	Shortage of geometric entities
0x0002	(ODH_SKIP):	Entity skipped due to high feed rate
0x0004	(ODH_OVERA):	Over-acceleration
0x0008	(ODH_DEFID):	Shortage of dynamic entities

RunSts	Mnemonic	Description
0x0001	(SWI_ONLYROTA):	Only rotary axes are moved
0x0100	CWI_ABOFIX	Canned cycles abort

Fholsts	Mnemonic	Description
0x8000	FHOLD_ON	FeedHold status on interpolator
0x4000	FHOLD_ONEMOV	FeedHold status at movement end
0x2000	FHOLD_INTERP	Internal FeedHold status
0x1000	FHOLD_WAIT	FeedHold status for Wait
0x0001	FAST_STOP	Fast stop requested
0x0002	FAST_MANLOP	Fast stop requested on manual limits
0x0004	FAST_MANHALT	Halt requested on manual limits

MovStat	Mnemonic	Description
0	MOV_ST_NULL	Non-existent movement
1	MOV_ST_STRT	Movement in starting phase
2	MOV_ST_IDLE	Movement in idle status
3	MOV_ST_RUN	Movement running
4	MOV_ST_HOLD	Movement in Hold
5	MOV_ST_STOP	Movement in stop
6	MOV_ST_EMER	Movement in emergency
7	MOV_ST_TOLL	Movement in axes tolerance pah

MovSub	Mnemonic	Description
0	MOV_ST_INIT	Movement initialization (MOV_ST_RUN)
1	MOV_ST_NORM	Movement running (MOV_ST_RUN)
2	MOV_ST_STOH	Movement in Hold stop (MOV_ST_STOP)
3	MOV_ST_STOR	Movement in Reset stop (MOV_ST_STOP)
4	MOV_ST_STOE	Movement in Emerg stop (MOV_ST_STOP)
5	MOV_ST_STOX	Movement in stop (MOV_ST_STOP)

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret      : dword;
IntInfo  : InterpInfo_Struct;
Ret := GMC_InterpReadInfo (1, IntInfo) ;
```

3.13 GMC_InterpReset

Function GMC_InterpReset asks a PLC interpolator to reset itself.

Syntax:

```
ret_code := GMC_Reset (ExecMode, ExecStatus, InterpId) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

InterpId (INT) Interpolator identifier PLC [1..40]

Execution mode:

Wait / NoWait

Use:

The interpolator specified is asked to reset itself so that it can restart for new operations. All movements underway are ended, and so are all the requests placed in a queue and not yet executed. This may also be used to release the interpolator from an error or an emergency condition.



This command is deemed to be over as soon as the interpolator has acquired the request, not when it has completed the reset stage. It is necessary to verify the status of the interpolator through function GMC_InterpreadInfo.

Return values

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160011	Interpolator type wrong (internal error)
0x00160014	Interpolator not present

Example

```
Ret      : dword;
InterpId : INT ;
...
Ret := GMC_InterpReset (MODE_WAIT, UL0, InterpId);
```

See also

GMC_InterpExitHold, GMC_EnterHold, GMC_InterpExitHold, GMC_ExitHold, GMC_Reset.

3.14 GMC_Reset

Function GMC_Reset asks all PLC interpolators to reset themselves.

Syntax:

```
ret_code := GMC_Reset (ExecMode, ExecStatus) ;
```

Input parameters

ExecMode (INT) Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]

ExecStatus (DWORD) Command execution status

Execution mode:

Wait / NoWait

Use

All interpolators are asked to reset themselves so that they will be able to restart for new operations. All movements underway are ended, and so are all the requests placed in a queue and not yet executed.

This call is also used to release the interpolators from an error or an emergency condition.



This command is deemed to be over as soon as the interpolators have acquired the request, not when the interpolators have completed the reset stage. It is necessary to verify the status of the interpolators through function GMC_InterpreadInfo.

Return values

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example

```
Ret : dword;  
Ret := GMC_Reset (MODE_WAIT, UL0);
```

See also

GMC_InterpExitHold, GMC_EnterHold, GMC_InterpExitHold, GMC_ExitHold, GMC_InterpReset.

3.15 GMC_Move

Function GMC_Move executes a linear movement.

Syntax:

```
ret_code := GMC_Move (ExecMode, ExecStatus, InterId, EndSignal, AxisId, Position) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	Interpolator identifier PLC [1..40]
<i>AxisId</i> (INT)	Axis identifier [1...64]
<i>Position</i> (LREAL)	Final position

Output parameters:

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (<i>EndSignal</i> set to TRUE)
-------------------------	---

Possible overload

GMC_Move(INT,DWORD,INT,BOOL,INT,LREAL[,INT,LREAL]...)

Execution mode

Wait / NoWait

Use:

This function permits the execution of a synchronous linear motion on one or more axes. At the end of movement execution, signal “*EndSignal*” is set to true. Parameters “*AxisId*” and “*Position*” may be duplicated making it possible to request the synchronous movement of several axes, up to 12.

The movement is executed at the feed rate and with all the acceleration/deceleration, jerk and minimum movement time dynamic parameters in fast mode, unless otherwise specified in the entry Feed structure. This structure programs values of speed, acceleration, deceleration and jerk which will be used in the movement. If these fields are not filled, the movement will be executed by default dynamics.

The following table describes the Feed Structure fields:

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed Acceleration (unit/s ²)
Decel	LREAL	Programmed Deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Speed calculated only for linear axis

Information below will be available from OPENcontrol release 2.1.

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

Below the values that can be adopted by the *TypeFeed* variable:

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Speed is to be intended as vector speed, that is to say the movement speed of all the axes together and not of the single axes.

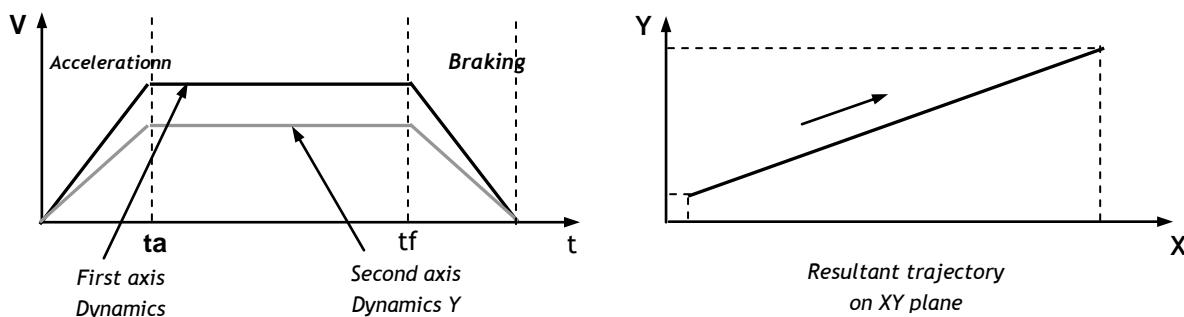
The “Position” parameter may take different meanings depending on the axis programming context. In the case of absolute programming, it specifies the end point where the axis will be at end of movement; in the case of incremental programming, it defines the distance (with sign) that the axis must cover starting from its current position. At any rate, all points are to be construed as relative to the origin activated on the axis programmed. Before a movement is executed, the relative operating limits, if active, are verified and may generate an Over travel error.



This function may be programmed from within a continuous stage.

Coordinating the axes of a movement

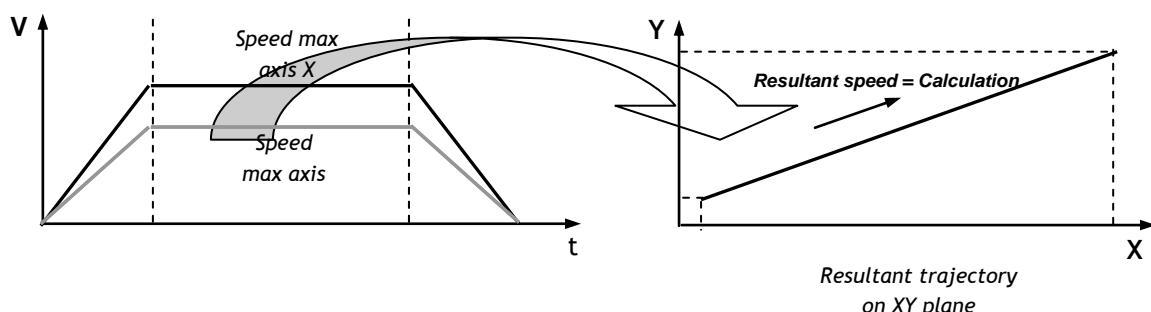
The axes defined in GMC_Move start and end their movements simultaneously; this means that they are “interpolated” with one another and hence the trajectory specified for the movement will be maintained. From the dynamic standpoint, each axis adopts a speed curve of its own, regardless of the other axes in terms of feed rate reached and acceleration and jerk values; since the motions of the various axes are mutually interpolated, the acceleration stage ends at the same time for all the axes: similarly, the deceleration stage starts simultaneously for all axes.



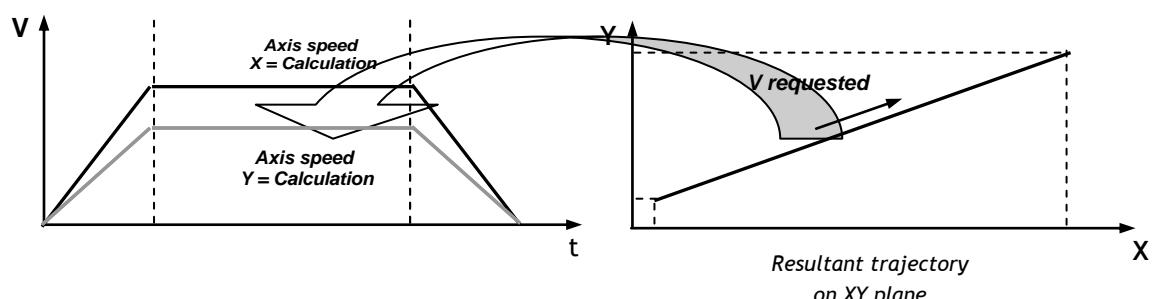
The system makes sure that the dynamic parameters requested for each axis do not exceed the (feed, acceleration and jerk) values programmed for it. If the dynamic values requested for an axis are exceedingly high, the system will limit the dynamics of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes having different clocks must be moved separately.

Programmed speed

As a rule, movements are performed at the highest possible feed rate, which is determined as a function of the maximum rate that each individual axis can reach. The speed obtained over a trajectory therefore depends both on the maximum feed that an axis can reach and on the extent to which an axis moves relative to global motion (ratio between the distance covered by the axis and the distance covered by the global movement).



Moreover, it is possible to set the “speed on the profile”, i.e., resultant speed on the global movement; in this case the speed at which an axis moves will be determined starting from the speed on the profile and the component of the movement of the individual axis relative to the overall distance to be covered “on the profile”.



Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated

0x0016000D	Interpolation activities exhausted
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(* Executes the movement of the axis with ID 1 at point 100.0 *)
Ret := GMC_Move (MODE_WAIT, ULO, InterpId, M0_0, Feed, 1, 100.0) ;

```

See also :

GMC_SetFeed, GMC_SetRamp, GMC_SetJerk, GMC_SetFeedOverride, GMC_AbsoluteQuote, GMC_IncrementalQuote, GMC_ActivateOrigin

3.16 GMC_MoveJog

Function GMC_MoveJog executes the manual movement in Jog.

Syntax:

```
ret_code := GMC_MoveJog (ExecMode, ExecStatus, Interpld, EndSignal, ManMode, AxisId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>ManMode</i> (WORD)	Execution mode
<i>AxisId</i> (INT)	Axis identifier [1...64]

Output parameter:

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (a TRUE)
-------------------------	--

Possible overload:

GMC_MoveJog(INT,DWORD,INT,BOOL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This function permits the execution of a manual motion in asynchronous Jog on one or more axes. At the end of movement, signal “*EndSignal*” is set to true. The parameter “*AxisId*” may be duplicated making it possible to request the asynchronous movement of up to 12 axes.

The “*ManMode*” parameter determines the manual movement which may be executed in normal way or subjected to the use of an hand wheel, the parameter is in bit code using the following mask:

ManMode	Mnemonic	Description
0x0001	MAN_HANDWHL	Movement through hand wheel
0x0002	MAN_HWLIMSTOP	Stops the axis on limits instead of setting the emergency status
0x0004	MAN_HWLIMALL	Stops all axes on limits instead of setting the emergency status

The movement is executed at the feed rate and with all the acceleration/deceleration, jerk and minimum movement time dynamic parameters in manual mode. Each axis is moved in asynchronous way compared to the others and the dynamics is managed through GMC_SetFeedOverride and GMC_SetFeedOverrideAll functions.

If software over travels have been activated and homing executed, working area will be traced by limits, otherwise an unlimited motion will be programmed. Motion stops only if the GMC_InterpReset interpolator is reset.



The function CAN'T be programmed within a continuous mode.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Movement elements have been used up
0x00160005	Too many axes involved
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities exhausted
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160036	EndSignal synchronism variable wrong
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Performs hand wheel move for axes ID 1 and ID 2 *)
Ret := GMC_MoveJog (MODE_WAIT, ULO, InterpId, M0_0, MAN_HANDWHEEL, 1, 2) ;

```

See also:

GMC_SetFeed, GMC_SetRamp, GMC_SetJerk, GMC_SetFeedOverride, GMC_HwndON, GMC_HwndOFF, GMC_HwndExe, GMC_HwndParam

3.17 GMC_MoveOnPosition

Function `GMC_MoveOnPosition` executes a linear movement conditional on the position of the axis.

Syntax:

```
ret_code := GMC_MoveOnPosition (ExecMode, ExecStatus, InterId, EndSignal,  
                                Mode, StartAxisId, StartAxisPos, AxisId, Position) ;
```

Input parameters:

<code>ExecMode</code> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<code>ExecStatus</code> (DWORD)	Command execution status
<code>InterId</code> (INT)	Interpolator identifier PLC [1..40]
<code>Mode</code> (INT)	Starting movement mode
<code>StartAxisId</code> (INT)	Axis ID [1...64] for starting synchronism
<code>StartAxisPos</code> (LREAL)	Axis position for starting synchronism
<code>AxisId</code> (INT)	Axis identifier [1...64]
<code>Position</code> (LREAL)	Final position

Output parameters

<code>EndSignal</code> (BOOL)	Boolean variable to give the end of activity signal (<code>EndSignal</code> set to TRUE)
-------------------------------	---

Possible overload

`GMC_MoveOnPosition(INT,DWORD,INT,BOOL,INT,INT,LREAL,INT,LREAL [,INT,LREAL]...)`

Execution mode:

Wait / NoWait

Use:

This function permits the execution of a synchronous linear movement on one or more axes, whose start is conditional on the position of an axis. At the end of movement execution, signal “`EndSignal`” is set to true. Parameters “`AxisId`” and “`Position`” may be duplicated making it possible to request the synchronous movement of several axes, up to 12.

The movement is activated when the “*StartAxisId*” axis meets the “*Mode*” condition relating to position “*Position*”, i.e.,:

Mode	Mnemonic	Description
10	CND_AXL	Starts when axis StartAxisId is at an interpolated position lower than Position. The position must take into account the offsets active on the StartAxisId axis.
11	CND_AXG	Starts when axis StartAxisId is at an interpolated position higher than or equal to Position. The position must take into account the offsets active on the StartAxisId axis.
12	CND_AXLABS	Starts when axis StartAxisId is at an interpolated position lower than Position. This position must be construed as absolute.
13	CND_AXGABS	Starts when axis StartAxisId is at an interpolated position higher than or equal to Position. This position must be construed as absolute.
14	CND_AXLph	Starts when axis StartAxisId is at a real position lower than Position. The position must take into account the offsets active on the StartAxisId axis.
15	CND_AXGph	Starts when axis StartAxisId is at a real position higher than or equal to Position. The position must take into account the offsets active on the StartAxisId axis.
16	CND_AXLABSph	Starts when axis StartAxisId is at a real position lower than Position. This position must be construed as absolute.
17	CND_AXGABSpj	Starts when axis StartAxisId is at a real position higher than or equal to Position. This position must be construed as absolute.

The movement is executed at the feed rate and with all the acceleration/deceleration, jerk and minimum movement time dynamic parameters in fast mode, unless otherwise specified by the structure at feed input. This structure defines the values of speed, deceleration and jerk which will be used in the movement. If these parameters are not set, the movement will be executed by default dynamics.

The following table describes the field of *Feed* structure:

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Following information will be also available with the OPENcontrol release 2.1.

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

Below the values that can be adopted by the *TypeFeed* variable:

<i>TypeFeed</i>	<i>Mnemonic</i>	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Speed is to be intended as vector speed, that is to say the movement speed of all the axes together and not of the single axes.

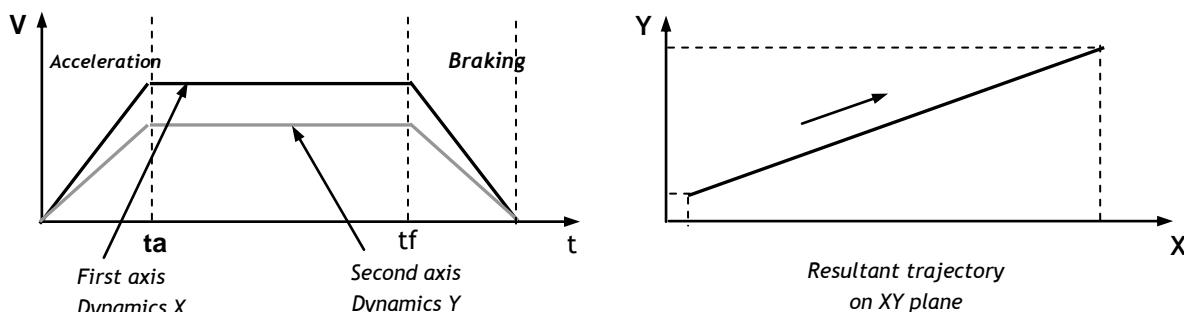
The “Position” parameter may take different meanings depending on the axis programming context. In the case of absolute programming, it specifies the end point where the axis will be at end of movement; in the case of incremental programming, it defines the distance (with sign) that the axis must cover starting from its current position. At any rate, all points are to be construed as relative to the origin activated on the axis programmed. Before a movement is executed, the relative operating limits, if active, are verified and may generate an Over travel error.



This function may be programmed from within the continuous stage. The movement will always start at nil feed to be able to execute the activation checks

Coordinating the axes of a movement

The axes defined in GMC_MoveOnPosition start and end their movements simultaneously; this means that they are “interpolated” with one another and hence the trajectory specified for the movement will be maintained. From the dynamic standpoint, each axis adopts a speed curve of its own, regardless of the other axes in terms of feed rate reached and acceleration and jerk values; since the motions of the various axes are mutually interpolated, the acceleration stage ends at the same time for all the axes:

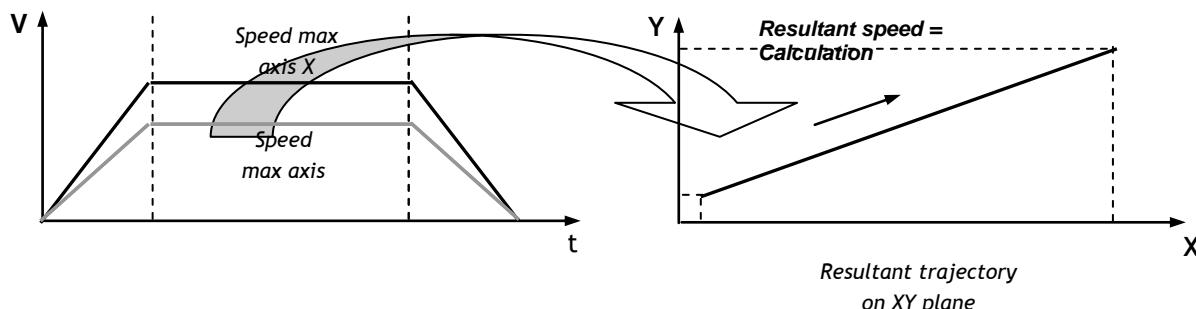


similarly, the deceleration stage starts simultaneously for all axes.

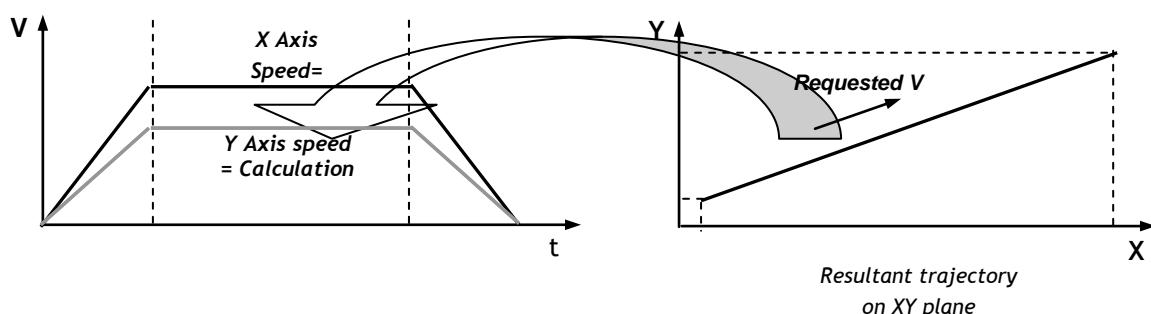
The system makes sure that the dynamic parameters requested for each axis do not exceed the (feed, acceleration and jerk) values programmed for it. If the dynamic values requested for an axis are exceedingly high, the system will limit the dynamics of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes having different clocks must be moved separately.

Programmed speed

As a rule, movements are performed at the highest possible speed rate, which is determined as a function of the maximum rate that each individual axis can reach. The speed obtained over a trajectory therefore depends both on the maximum speed that an axis can reach and on the extent to which an axis moves relative to global motion (ratio between the distance covered by the axis and the distance covered by the global movement).



Moreover, it is possible to set the “speed on the profile”, i.e., resultant speed on the global movement; in this case the speed at which an axis moves will be determined starting from the speed on the profile and the component of the movement of the individual axis relative to the overall distance to be covered “on the profile”.



Return values:

The following table lists possible values of *ret_code* variable

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities exhausted
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch

0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160032	Starting movement condition wrong
0x0016005E	Rollover axis position wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(* Executes axis movement with ID 1 to dimension 100.0 when the axis
   with Id 2 has a position higher than 200.0*)
Ret := GMC_MoveOnPosition (MODE_WAIT, ULO, InterpId, M0_0, CND_AXG, 2,
                           200.0, 1, 100.0) ;

```

See also:

[GMC_SetJerk](#), [GMC_SetRamp](#), [GMC_SetFeedOverride](#), [GMC_AbsoluteQuote](#), [GMC_IncrementalQuote](#), [GMC_ActivateOrigin](#)

3.18 GMC_MoveOnSignal

Function `GMC_MoveOnSignal` executes a linear movement conditional on the signal status.

Syntax:

```
ret_code := GMC_MoveOnSignal (ExecMode, ExecStatus, InterpId, EndSignal,
                             IniSignal, Mode, Feed, AxisId, Position) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>IniSignal</i> (BOOL)	Boolean variable for starting synchronism
<i>Mode</i> (INT)	Starting movement mode
<i>Feed</i> (FeedDescr_Struct)	Movement speed
<i>AxisId</i> (INT)	Axis identifier [1...64] for max 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters:

EndSignal (BOOL) Boolean variable to give the end of activity signal (a TRUE)

Possible overload:

`GMC_MoveOnSignal (INT,DWORD,INT,BOOL,BOOL,INT, FeedDescr_Struct INT,LREAL[,INT,LREAL]...)`

Execution mode:

Wait / NoWait

Use:

This function permits the execution of a synchronous linear movement on one or more axes, whose start is conditional on the signal status. At the end of movement execution, signal “*EndSignal*” is set to true. Parameters “*AxisId*” and “*Position*” may be duplicated making it possible to request the synchronous movement of several axes, up to 12.

The movement is activated when the “*IniSignal*” axis meets the “*Mode*” condition:

Mode	Mnemonic	Description
1	CND_RAISE	Start on rising edge of signal <i>IniSignal</i>
2	CND_FAIL	Start on falling edge of signal <i>IniSignal</i>
3	CND_ON	Start on “ON” level (true) of signal <i>IniSignal</i>
4	CND_OFF	Start on “OFF” level (false) of signal <i>IniSignal</i>

The movement is executed at the feed rate and with all the acceleration/deceleration, jerk and minimum movement time dynamic parameters in fast mode, unless otherwise specified in the structure of the input feed. This structure defines speed values, acceleration, deceleration and jerk which will be used in the movement. If these parameters are not set, the movement will be executed by default dynamics.

The following table defines the fields of the Feed structure:

Fields	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Following information will be also available with the OPENcontrol release 2.1.

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

Here follows the values that can be adopted by the *TypeFeed* variable.

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Speed is to be intended as vector speed, that is to say the movement speed of all the axes together and not of the single axes.

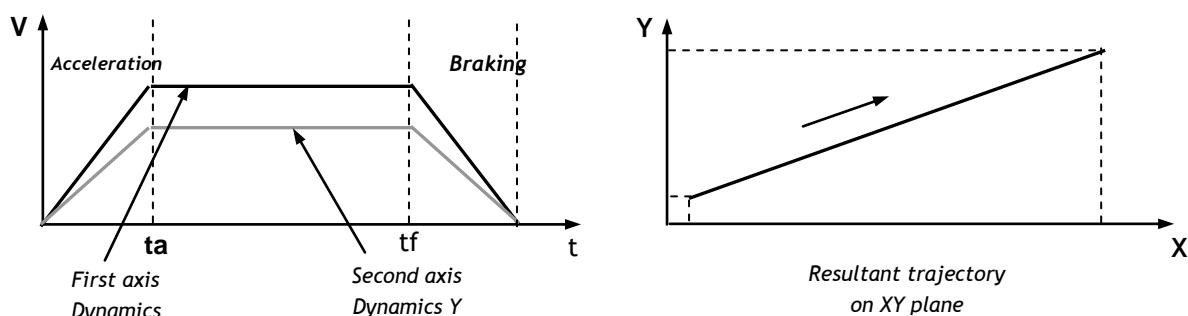
The “Position” parameter may take different meanings depending on the axis programming context. In the case of absolute programming, it specifies the end point where the axis will be at end of movement; in the case of incremental programming, it defines the distance (with sign) that the axis must cover starting from its current position. At any rate, all points are to be construed as relative to the origin activated on the axis programmed. Before a movement is executed, the relative operating limits, if active, are verified and may generate an Over travel error.



This function may be programmed from within the continuous stage. The movement will always start at nil feed to be able to execute the activation checks

Coordinating the axes of a movement

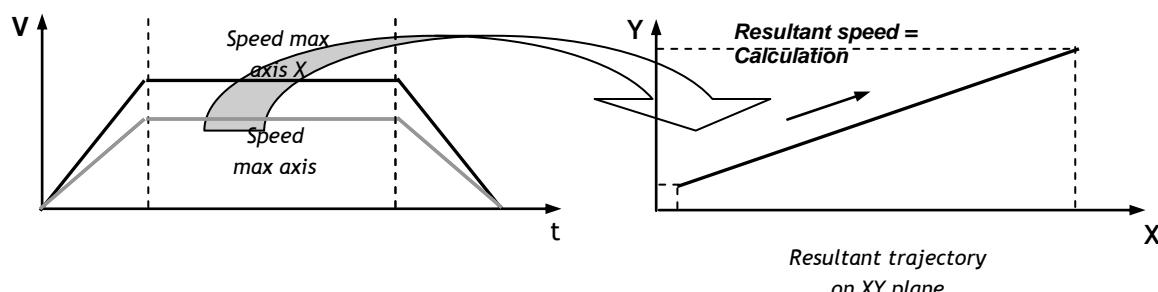
The axes defined in GMC_MoveOnSignal start and end their movements simultaneously; this means that they are “interpolated” with one another and hence the trajectory specified for the movement will be maintained. From the dynamic standpoint, each axis adopts a speed curve of its own, regardless of the other axes in terms of feed rate reached and acceleration and jerk values; since the motions of the various axes are mutually interpolated, the acceleration stage ends at the same time for all the axes: similarly, the deceleration stage starts simultaneously for all axes.



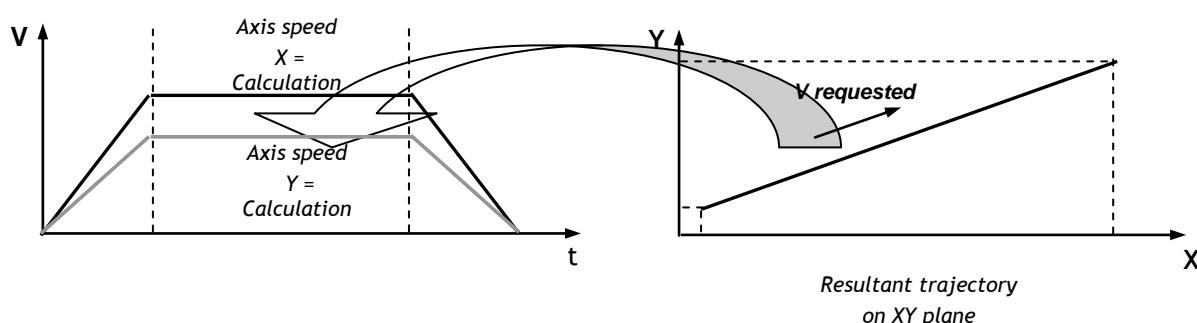
The system makes sure that the dynamic parameters requested for each axis do not exceed the (feed, acceleration and jerk) values programmed for it. If the dynamic values requested for an axis are exceedingly high, the system will limit the dynamics of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes having different clocks must be moved separately.

Programmed feed

As a rule, movements are performed at the highest possible feed rate, which is determined as a function of the maximum rate that each individual axis can reach. The speed obtained over a trajectory therefore depends both on the maximum feed that an axis can reach and on the extent to which an axis moves relative to global motion (ratio between the distance covered by the axis and the distance covered by the global movement).



Moreover, it is possible to set the “speed on the profile”, i.e., resultant speed on the global movement; in this case the speed at which an axis moves will be determined starting from the speed on the profile and the component of the movement of the individual axis relative to the overall distance to be covered “on the profile”.



Return values:

The following table lists all possible values that *ret_code* can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities exhausted
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160046	Null speed
0x00160036	EndSignal or IniSignal synchronism variable wrong
0x00160032	Starting movement condition wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(*Executes axis movement with ID 1 to dimension 100.0 when M0_1
   is set to TRUE status *)
Ret := GMC_MoveOnSignal (MODE_WAIT, UL0, InterpId, M0_0, M0_1, SG_ON,
                        Feed, 1, 100.0) ;

```

See also:

[GMC_SetRamp](#), [GMC_SetJerk](#), [GMC_SetFeedOverride](#), [GMC_AbsoluteQuote](#), [GMC_IncrementalQuote](#), [GMC_ActivateOrigin](#)

3.19 GMC_MoveUntilSignal

Function **GMC_MoveUntilSignal** executes linear movement ending on signal.

Syntax:

```
ret_code := GMC_MoveUntilSignal (ExecMode, ExecStatus.InterId, Touched, StopSignal,
                                 VelMode, StopDist, StopMode, AxisId, Position) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	Interpolator identifier PLC [1..40]
<i>StopSignal</i> (BOOL)	Boolean variable to be used for movement stop
<i>VelMode</i> (INT)	Feed determination mode
<i>StopDist</i> (LREAL)	Safe stopping distance
<i>StopMode</i> (INT)	Movement stop signal management mode
<i>AxisId</i> (INT)	Axis identifier [1..64] for max 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters:

<i>Touched</i> (BOOL)	Boolean variable to give the end of activity signal (<i>EndSignal</i> set to TRUE)
-----------------------	---

Possible overload:

GMC_MoveUntilSignal (INT,DWORD,INT,BOOL, BOOL, INT, LREAL, INT, INT, LREAL [,INT,LREAL]...)

Execution mode:

Wait / NoWait

Use:

This function permits the execution of a synchronous linear movement on one or more axes, a movement whose end is conditional on the status of a signal. At end of movement execution, signal «*Touched*» is set to true only if the movement has been terminated due to the stop condition having occurred. Parameters “*AxisId*” and “*Position*” may be duplicated in order to request synchronous movement of several axes, up to 12.

The movement is terminated when signal «*StopSignal*» meets the «*StopMode*» condition, i.e.:

StopMode	Mnemonic	Description
1	CND_RAISE	Stop on rising edge of signal StopSignal
2	CND_FAIL	Stop on falling edge of signal StopSignal
3	CND_ON	Stop on “ON” level (true) of signal StopSignal
4	CND_OFF	Stop on “OFF” level (false) of signal StopSignal

Feed rates conditioned by the «VelMode» parameter may be calculated according to two different modalities, that is to say:

VelMode	Mnemonic	Description
0	MODE_FAST	Movement in fast mode
1	MODE_STOP	Movement with V calculated so that the movement can be stopped within StopDist distance

In the case of MODE_FAST the movement is executed at the feed rate and with all acceleration/deceleration, jerk and minimum movement time dynamic parameters in fast mode, unless otherwise specified by the structure at Feed input. This structure defines speed, acceleration, deceleration and jerk values which will be used during movement. If these values are not set, the movement will be executed by default dynamics.

The following table defines the fields of the Feed structure.

Fields	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Following information will be available from OPENcontrol release 2.1:

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

Here follows the values that can be adopted by the TypeFeed variable.

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

In this case the distance covered between the time the stop event occurs and the time the movement stops is a function of the feed at which the axes were moving when the stop event occurred.

Since the movement is executed in fast mode, the distance may be estimated to be of the order of several mm.

For example, a linear ramp having been set, with a feed of 30000 mm/min and a deceleration of 1500 mm/sec², stopping distance will be ca 83mm, calculated as follows:

$$\text{Space} = \frac{V}{2 A}$$

In the case of MODE_STOP feed is calculated so as to ensure that the movement stops over a distance corresponding to the one required by parameter «StopDist». In this case, the formula applied is the reverse of the one used for linear ramps :

$$F = \sqrt{2 A * StopDist}$$

In any event, feed rate must be construed as a vector quantity, i.e., as the speed of displacement of the entire set of axes moved, not of the individual axes.

The “Position” parameter may take different meanings depending on the axis programming context. In the case of absolute programming, it specifies the end point where the axis will be at end of movement; in the case of incremental programming, it defines the distance (with sign) that the axis must cover starting from its current position. At any rate, all points are to be construed as relative to the origin activated on the axis programmed. Before a movement is executed, the relative operating limits, if active, are verified and may generate an Over travel error.



This function may be programmed from within the continuous stage and will behave in different ways depending on whether the stop condition is verified

Stop condition NOT verified (*Touched* a false)

Continuous stage continues regularly.

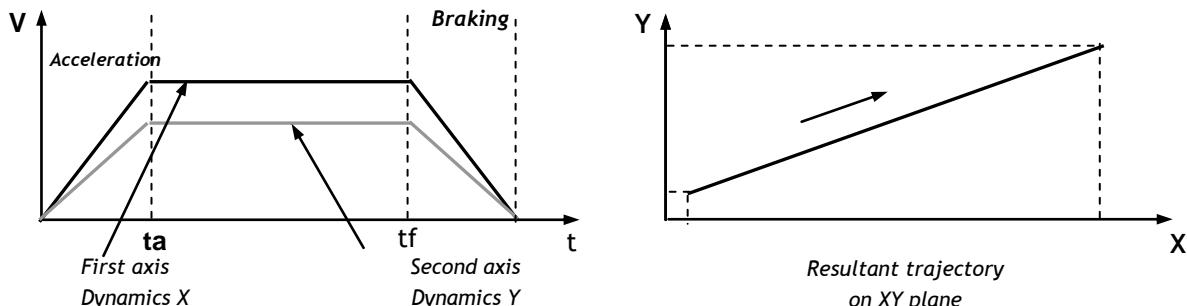
Stop condition verified (*Touched* a true)

The system will end the continuous stage independently (all movements calculated so far will be deleted) and will indicate the fact in an ad hoc bit of the continuous stage control status word. At this point, two conditions may occur:

- Continuous stage already closed by GMC_ContinuousEnd instruction: all movements programmed in continuous mode are cancelled.
- Continuous stage not yet closed by GMC_ContinuousEnd: the system is going to ignore all movement requests until a continuous stage closure instruction is received.

Coordinating the axes of a movement

The axes defined in GMC_MoveOnSignal start and end their movements simultaneously; this means that they are “interpolated” with one another and hence the trajectory specified for the movement will be maintained. From the dynamic standpoint, each axis adopts a speed curve of its own, regardless of the other axes in terms of feed rate reached and acceleration and jerk values; since the motions of the various axes are mutually interpolated, the acceleration stage ends at the same time for all the axes:

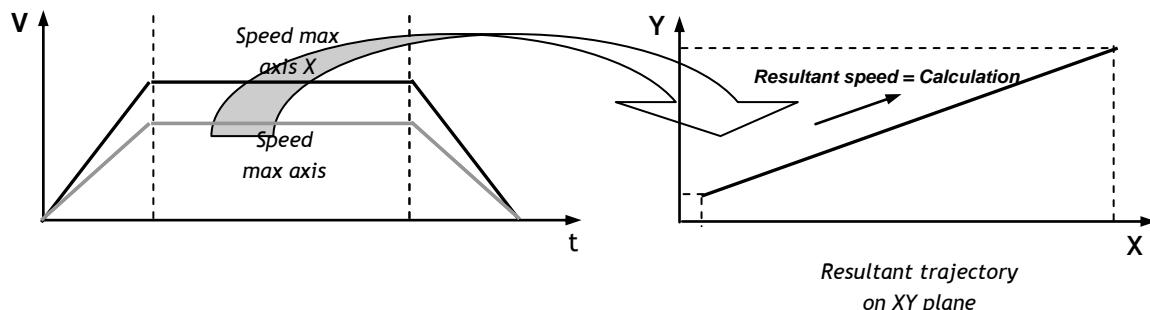


similarly, the deceleration stage starts simultaneously for all axes.

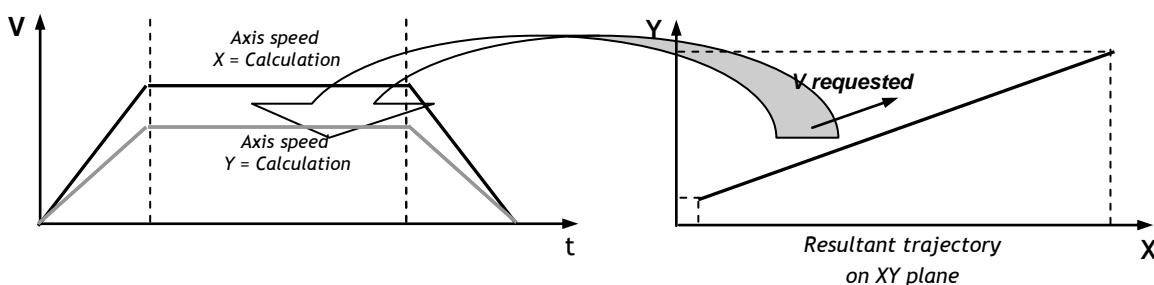
The system makes sure that the dynamic parameters requested for each axis do not exceed the (feed, acceleration and jerk) values programmed for it. If the dynamic values requested for an axis are exceedingly high, the system will limit the dynamics of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes having different clocks must be moved separately.

Programmed feed

As a rule, movements are performed at the highest possible feed rate, which is determined as a function of the maximum rate that each individual axis can reach. The speed obtained over a trajectory therefore depends both on the maximum feed that an axis can reach and on the extent to which an axis moves relative to global motion (ratio between the distance covered by the axis and the distance covered by the global movement).



Moreover, it is possible to set the “speed on the profile”, i.e., resultant speed on the global movement; in this case the speed at which an axis moves will be determined starting from the speed on the profile and the component of the movement of the individual axis relative to the overall distance to be covered “on the profile”.



Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities exhausted
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160046	Null speed
0x00160036	EndSignal or StopSignal synchronism variable wrong
0x0016005E	Rollover axis position wrong
0x00160037	VelMode parameter wrong
0x00160034	Movement stop condition wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(*Executes movement of axis with ID 1 at point 100.0, stopping on rising
 edge of M0_1 and notifying stop on M0_0  *)
Ret := GMC_MoveUntilSignal (MODE_WAIT, ULO, InterpId, M0_0, M0_1,
                           MODE_FAST, 0.0, CND_RAISE, Feed, 1, 100.0);

```

See also:

GMC_SetRamp, GMC_SetJerk, GMC_SetFeedOverride, GMC_AbsoluteQuote, GMC_IncrementalQuote, GMC_ActivateOrigin

3.20 GMC_Poly

GMC_Poly function executes a polynomial interpolation.

Syntax

```
ret_code := GMC_Poly (ExecMode, ExecStatus, InterpId, EndSignal, Feed, AxisId, Poly) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>Length</i> (LREAL)	Polynomial length
<i>Feed</i> (FeedDescr_Struct)	Feed motion
<i>AxisId</i> (INT)	Axis identifier [1...64] for up to 12 axes
<i>Poly</i> (ARRAY[0..5] OF LREAL)	Polynomial coefficient

Output parameters

<i>EndSignal</i> (BOOL)	Boolean variable on which is indicated (at TRUE) the end of activity
-------------------------	--

Possible overloads

GMC_Poly (INT,DWORD,INT,BOOL, FeedDescr_Struct,
INT, ARRAY[0..5] OF LREAL[,INT, ARRAY[0..5] OF LREAL]...)

Execution mode

Wait / NoWait

Use

This function executes a polynomial interpolation using a fifth degree polynomial having length and coefficients set. When motion finishes, the “*EndSignal*” signal is set as true. “*AxisId*” and “*Poly*” parameters can be duplicated to require the synchronous motion for up to 12 axes (at least 2).

The motion is executed at rapid feed or at feed defined by the “*Feed*” structure value. This structure defines feed, acceleration, deceleration and jerk values that will be used in the motion. If this fields are not completed, the motion will be executed through a default dynamics.

The following table describes Feed structure fields.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Acceleration (unit/s ²)
Decel	LREAL	Deceleration (unit/s ²)
Jerk	LREAL	Jerk in acceleration (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated for linear axes only

Following information will be available from OPENcontrol release 2.1.

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

The following table lists all the possible *TypeFeed* values:

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Velocity has to be intended as vector speed, which is the feed rate group of the moved axes and not of the single axes.

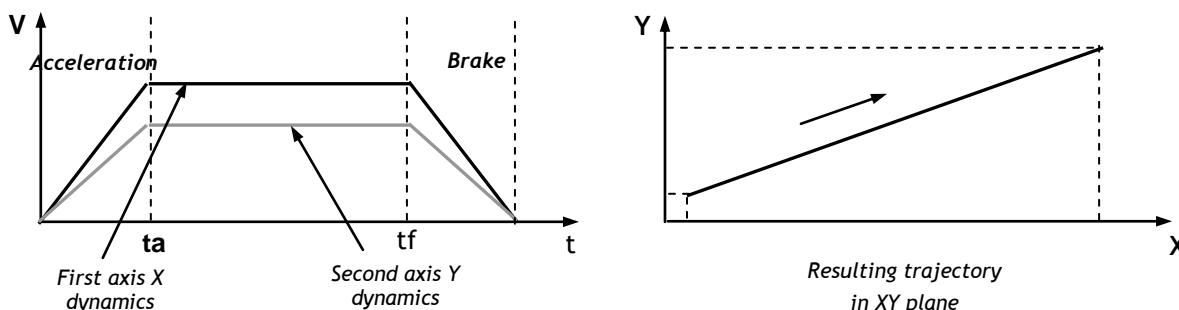
“Length” parameter is the interpolating polynomial length, while “Poly” parameter is an array containing polynomial coefficients from the known end to the maximum degree end. The control will calculate the final point according to the programming type (absolute or incremental). If operative limits are enabled, they will be checked before the movement execution, possibly creating an Over travel error.



Function can be programmed within a continuous

Coordinating the axes of a movement

The axes defined in GMC_Poly *start and end their movements simultaneously*; this means that they are each other “interpolated” and hence the trajectory specified for the movement will be maintained. From the dynamic standpoint, each axis adopts a speed curve of its own, regardless the other axes in terms of feed rate reached and acceleration and jerk values; since the motions of the various axes are mutually interpolated, the acceleration stage ends at the same time for all the axes

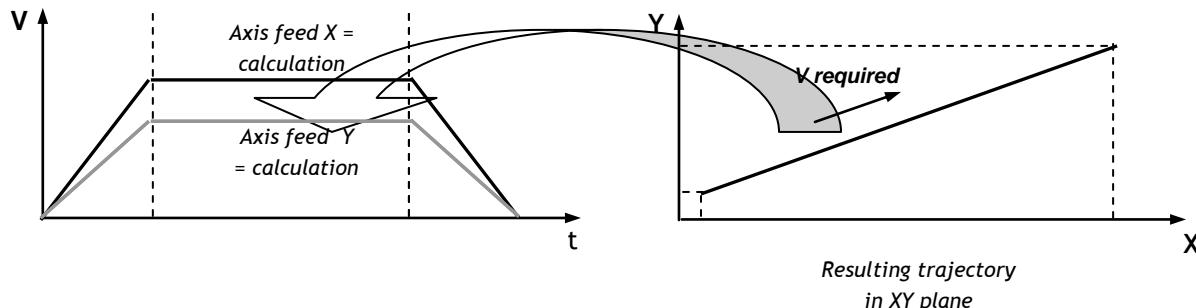


The system makes sure that the dynamic parameters requested for each axis do not exceed the (feed, acceleration and jerk) values programmed for it. If the dynamic values requested for an axis are exceedingly high, the system will limit the dynamics of the other axes involved in the movement. All axes involved in a movement must have the same servo clock; axes having different clocks must be moved separately.

Programmed feed

This function works using the "feed on profile", that is the feed resulting out of the global motion; in this case, the axis feed will be calculated starting from the profile feed rate and from the motion component of the single axis compared to the global distance "on profile" to run.

As a rule, movements are performed at the highest possible feed rate, which is determined as a function of the maximum rate that each individual axis can reach. The speed obtained over a trajectory therefore depends both on the maximum feed that an axis can reach and on the extent to which an axis moves relative to global motion (ratio between the distance covered by the axis and the distance covered by the global movement).



Return values

The following table lists all the possible values of *ret_code*:

<i>ret_code (HEX)</i>	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Motion elements expired
0x00160005	Too many axes involved
0x0016002C	Invalid axis id
0x0016002B	Axis id not found
0x0016002D	Gantry axis id
0x00160027	Spindle axis id
0x0016002F	Duplicated axis id
0x0016000D	Interpolation activities finished
0x00160028	Axes already used by an interpolator
0x00160029	Axes already used by an interpolator in Hold status
0x00160030	Axes already used as slave by an interpolator
0x0016002A	Axes having axes with different clocks
0x0016003C	Axis exceeds the positive Software limit
0x0016003D	Axis exceeding the negative Software limit

0x00160040	Axis on positive over travel
0x00160041	Axis on negative over travel
0x00160046	Null feed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x001600D4	Wrong polynomial starting point

Example

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
Ax_Poly1 : ARRAY[0..5] OF LREAL;
Ax_Poly2 : ARRAY[0..5] OF LREAL;
Ax_Poly3 : ARRAY[0..5] OF LREAL;
Length   : LREAL;

...
Feed.Value    := 10000.0 ;
Feed.TypeFeed := MODE_WORK;

Length      := 1.366138679e+001;
Ax_Poly1[0]   := 7.069500000e+001;
Ax_Poly1[1]   := -2.181330524e-001;
Ax_Poly1[2]   := -5.466496669e-016;
Ax_Poly1[3]   := -2.206548736e-006;
Ax_Poly1[4]   := 2.368758461e-006;
Ax_Poly1[5]   := -1.862318832e-007;

Ax_Poly2[0]   := 5.338700000e+001;
Ax_Poly2[1]   := 9.491715739e-001;
Ax_Poly2[2]   := 2.034070713e-016;
Ax_Poly2[3]   := -1.522183539e-006;
Ax_Poly2[4]   := 3.153126755e-006;
Ax_Poly2[5]   := -2.594548691e-007;

Ax_Poly3[0]   := -9.490000000e+001;
Ax_Poly3[1]   := -2.269169337e-001;
Ax_Poly3[2]   := -8.385639101e-016;
Ax_Poly3[3]   := 8.942373369e-006;
Ax_Poly3[4]   := 6.657105076e-006;
Ax_Poly3[5]   := -6.470537605e-007;

ret := GMC_Poly(MODE_WAIT, UL0, InterpId, M0_0, Feed,, Length , Feed ,
                 1, Ax_Poly1,2, Ax_Poly2,3, Ax_Poly3);

```

See also

[GMC_SetRamp](#), [GMC_SetJerk](#), [GMC_SetFeedOverride](#), [GMC_AbsoluteQuote](#), [GMC_IncrementalQuote](#), [GMC_ActivateOrigin](#)

3.21 GMC_ProtectAreaCmd

The GMC_ProtectAreaCmd function works on a protected area previously defined.

Syntax

```
ret_code:= GMC_ProtectAreaCmd (ExecMode, ExecStatus, InterpId, IdArea, Command);
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>IdArea</i> (INT)	Protected area identifier
<i>Command</i> (INT)	Command

Execution mode

Wait / NoWait

Use

Using this function it is possible to work on a protected area previously defined which id is known as *IdArea* (see GMC_ProtectAreaDefine). On this area, the following operations can be executed:

Command	Mnemonic	Description
1	PROT_ON_INT	Internal type of the protected area enabled. Motions have to be executed inside this area
2	PROT_ON_EXT	External type of the protected area enabled. Motions have to be executed outside the area
3	PROT_OFF	Protected area disabled. The area remains still defined
4	PROT_DEL	Protected area deleted. The definition is also cancelled

The protected area will be managed for all axes motions made by PLC and also for Process manual motions.

Return values

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160080	Undefined protected area
0x00160081	Wrong command on protected area

Example

```
Ret      : dword;
InterpId : INT;
AreaId   : INT;
...
(* Enables a protected area: it can move only externally *)
Ret := GMC_ProtectAreaCmd (MODE_WAIT, UL0, InterpId, AreaId, PROT_ON_EXT);
```

See also: GMC_ProtectAreaCmd

3.22 GMC_MoveRoundOFF

Function `GMC_MoveRoundOFF` executes a linear movement with “release”.

Syntax:

```
ret_code := GMC_MoveRoundOFF (ExecMode, ExecStatus, InterpId, EndSignal,
                               AxisId, Position, RoundPosition) ;
```

Input parameters:

<code>ExecMode</code> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<code>ExecStatus</code> (DWORD)	Command execution status
<code>InterpId</code> (INT)	Interpolator identifier PLC [1..40]
<code>AxisId</code> (INT)	Axis identifier [1...64] for max 12 axes
<code>Position</code> (LREAL)	Final position
<code>RoundPosition</code> (LREAL)	Round position

Output parameters:

<code>EndSignal</code> (BOOL)	Boolean variable to give the end of activity signal (<code>EndSignal</code> set to TRUE)
-------------------------------	---

Possible overload:

`GMC_MoveRoundOFF` (INT,DWORD,INT,BOOL, INT,LREAL,LREAL [,INT,LREAL,LREAL]...)

Execution mode

Wait / NoWait

Use:

This function permits the execution of a synchronous linear movement on one or more axes. Reaching or exceeding position “`RoundPosition`”, by all the axes defined in the movement, will be communicated through signal “`EndSignal`” which is set to true. Position “`RoundPosition`” - based on the direction of movement - must precede position “`Position`”. Parameters “`AxisId`”, “`Position`” and “`RoundPosition`” may be duplicated in order to request synchronous movement of several axes, up to 12.

If recalled within a “continuous stage” as the “`RoundPosition`” positions are reached, this function will start the movement, whether or not the starting conditions of the continuous stage have not yet been met. The movement is executed at the feed rate and with all acceleration/deceleration, jerk and minimum movement time dynamic parameters in fast mode, unless otherwise specified by the structure at Feed input. This structure defines speed, acceleration, deceleration and jerk values which will be used in the movement. If these parameters are not set, the movement will be executed by default dynamics.

The following table defines the fields of the Feed structure.

Fields	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Here follows the values that can be adopted by the *TypeFeed* variable.

- | | |
|--------------------------|---|
| TypeFeed = 0: MODE_RAPID | Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements. |
| 1: MODE_WORK | Movements will use the programmed speed and the dynamic parameters of programmed or work movement |
| 2: MODE_FEED_t | Movements are programmed in time and will use the dynamic parameters of work or programmed movements. |
| 3: MODE_FEED_1T | Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements. |

Speed is to be intended as vector speed, that is to say the movement speed of all the axes together and not of the single axes.

The “Position” and “RoundPosition” parameters may take different meanings depending on the axis programming context. In the case of absolute programming, it specifies the end point where the axis will be at end of movement and the round point; in the case of incremental programming, it defines the distance (with sign) that the axis must cover starting from its current position and the signal that the round dimension has been reached. At any rate, all points are to be construed as relative to the origin activated on the axis programmed. Before a movement is executed, the relative operating limits, if active, are verified and may generate an Over travel error.

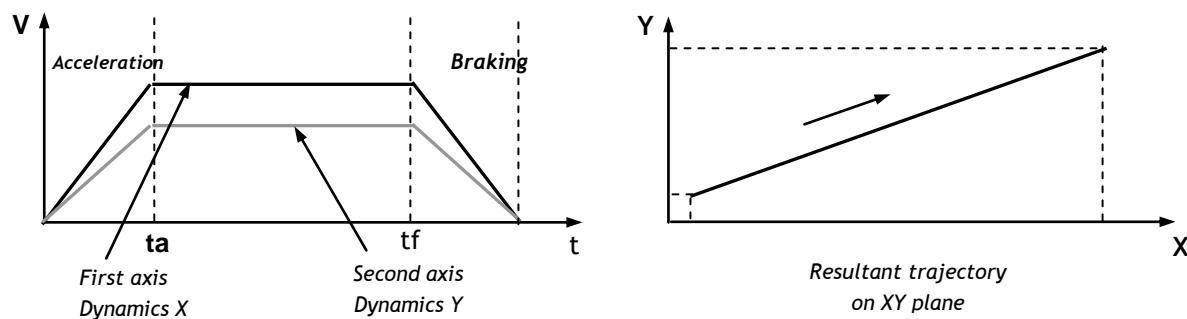


This function may be programmed from within the continuous stage.

Coordinating the axes of a movement

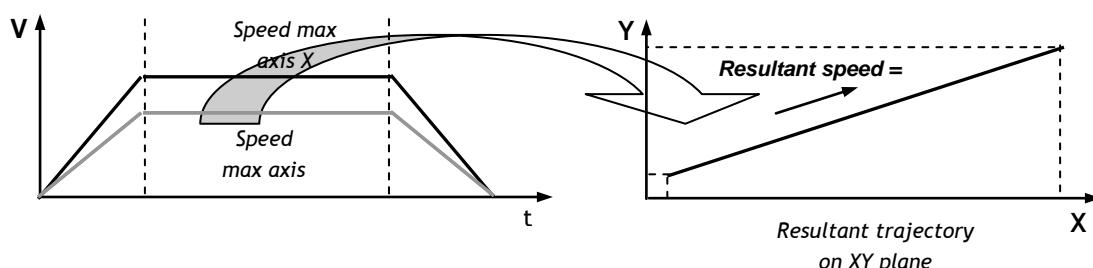
The axes defined in GMC_MoveRoundOff start and end their movements simultaneously; this means that they are “interpolated” with one another and hence the trajectory specified for the movement will be maintained. From the dynamic standpoint, each axis adopts a speed curve of its own, regardless of the other axes in terms of feed rate reached and acceleration and jerk values; since the motions of the various axes are mutually interpolated, the acceleration stage ends at the same time for all the axes: similarly, the deceleration stage starts simultaneously for all axes.

The system makes sure that the dynamic parameters requested for each axis do not exceed the (feed, acceleration and jerk) values programmed for it. If the dynamic values requested for an axis are exceedingly high, the system will limit the dynamics of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes having different clocks must be moved separately.

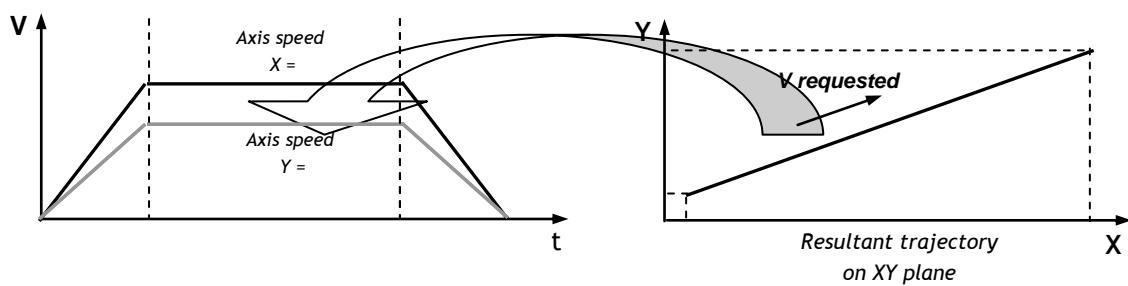


Programmed feed

As a rule, movements are performed at the highest possible feed rate, which is determined as a function of the maximum rate that each individual axis can reach. The speed obtained over a trajectory therefore depends both on the maximum feed that an axis can reach and on the extent to which an axis moves relative to global motion (ratio between the distance covered by the axis and the distance covered by the global movement).



Moreover, it is possible to set the “speed on the profile”, i.e., resultant speed on the global movement; in this case the speed at which an axis moves will be determined starting from the speed on the profile and the component of the movement of the individual axis relative to the overall distance to be covered “on the profile”.



Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities exhausted
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x0016005E	Rollover axis position wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(*Executes movement of axis with ID 1 at point 100.0, indicating M0_1
when there is 10 mm to go to the end of the movement *)
Ret := GMC_MoveRoundOFF (MODE_WAIT, UL0, InterpId, M0_1, Feed, 1, 100.0, 90.0);

```

See also:

GMC_SetRamp, GMC_SetJerk, GMC_SetFeedOverride, GMC_AbsoluteQuote, GMC_IncrementalQuote, GMC_ActivateOrigin

3.23 GMC_MoveUntilVariable

Function `GMC_MoveUntilVariable` executes a linear movement that ends on the basis of the value of a variable.

Syntax:

```
ret_code := GMC_MoveUntilVariable (ExecMode, ExecStatus, Interpld, Touched, StopVar,
                                   VelMode, StopDist, VarValue, StopMode Feed, AxisId,
                                   Position) ;
```

Input parameters:

<code>ExecMode (INT)</code>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<code>ExecStatus (DWORD)</code>	Command execution status
<code>Interpld (INT)</code>	Interpolator identifier PLC [1..40]
<code>StopVarl (INT)</code>	Integer variable to be used for movement stop
<code>VelMode (INT)</code>	Feed determination mode
<code>StopDist (LREAL)</code>	Safe stopping distance
<code>VarValue (INT)</code>	Test value for movement stop variable
<code>StopMode (INT)</code>	Movement stop signal management mode
<code>Feed (FeedDescr_Struct)</code>	Movement speed
<code>AxisId (INT)</code>	Axis identifier [1...64] for max 12 axes
<code>Position (LREAL)</code>	Final position

Output parameters:

<code>Touched (BOOL)</code>	Boolean variable to give the end of activity signal (a TRUE)
-----------------------------	--

Possible overload:

`GMC_MoveUntilSignal (INT,DWORD,INT,BOOL, INT,INT,LREAL,INT,INT, FeedDescr_Struct, INT,LREAL
[,INT,LREAL]...)`

Execution mode:

Wait / NoWait

Use:

This function permits the execution of a synchronous linear movement on one or more axes, a movement whose end is conditional on the status of a signal. At end of movement execution, signal «`Touched`» is set to true only if the movement has been terminated due to the stop condition having occurred. Parameters “`AxisId`” and “`Position`” may be duplicated in order to request synchronous movement of several axes, up to 12.

The movement finishes when variable «*StopVar*» meets the «*StopMode*» condition, i.e.:

StopMode =	40 CND_VRLESS	End if StopVar value is lower than VarValue
	41 CND_VRGREAT	End if StopVar value is higher than VarValue
	42 CND_VREQUAL	End if StopVar value is equal to VarValue
	43 CND_VRDIFF	End if StopVar value is different from VarValue

Feed rates conditioned by the «*VelMode*» parameter may be calculated according to two different modalities, that is to say:

VelMode =	0 MODE_FAST	Movement in fast mode
	1 MODE_STOP	Movement with V calculated so that the movement can be stopped within StopDist distance

In the case of MODE_FAST the movement is executed at the feed rate and with all acceleration/deceleration, jerk and minimum movement time dynamic parameters in fast mode, unless otherwise specified by the structure at Feed input. This structure defines speed, acceleration, deceleration and jerk values which will be used during movement. If these values are not set, the movement will be executed by default dynamics.

The following table defines the fields of the Feed structure.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Here follows the values that can be adopted by the *TypeFeed* variable:

TypeFeed =	0: MODE_RAPID	Movements will use the fast speed rate and the dynamic parameters of the rapid or programmed movements.
	1: MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
	2: MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
	3: MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

In this case the distance covered between the moment the stop event occurs and the moment the movement stops is a function of the feed at which the axes were moving when the stop event occurred. Since the movement is executed in fast mode, the distance may be estimated to be of the order of several mm.

For example, a linear ramp having been set, with a feed of 30000 mm/min and a deceleration of 1500 mm/sec², stopping distance will be ca 83mm, calculated as follows:

$$\text{Space} = \frac{V}{2 A}$$

In the case of MODE_STOP feed is calculated so as to ensure that the movement stops over a distance corresponding to the one required by parameter «StopDist». In this case, the formula applied is the reverse of the one used for linear ramps

$$V = \sqrt{2 A * StopDist}$$

In any event, feed rate must be construed as a vector quantity, i.e., as the speed of displacement of the entire set of axes moved, not of the individual axes.

The “Position” parameter may take different meanings depending on the axis programming context. In the case of absolute programming, it specifies the end point where the axis will be at end of movement; in the case of incremental programming, it defines the distance (with sign) that the axis must cover starting from its current position. At any rate, all points are to be construed as relative to the origin activated on the axis programmed. Before a movement is executed, the relative operating limits, if active, are verified and may generate an Over travel error.



This function may be programmed from within the continuous stage and will behave in different ways depending on whether the stop condition is verified

Stop condition NOT verified (*Touched* a false)

Continuous stage continues regularly.

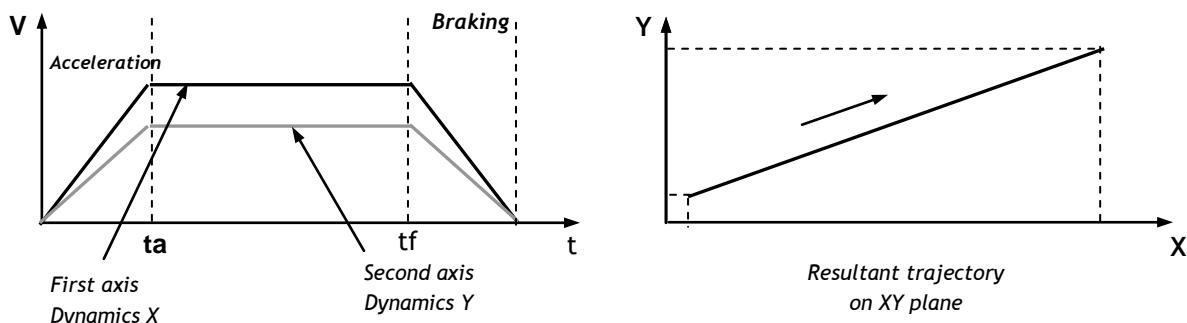
Stop condition verified (*Touched* a true)

The system will end the continuous stage independently (all movements calculated so far will be deleted) and will indicate the fact in an ad hoc bit of the continuous stage control status word. At this point, two conditions may occur:

- Continuous stage already closed by GMC_ContinuousEnd instruction: all movements programmed in continuous mode are cancelled.
- Continuous stage not yet closed by GMC_ContinuousEnd: the system is going to ignore all movement requests until a continuous stage closure instruction is received

Coordinating the axes of a movement

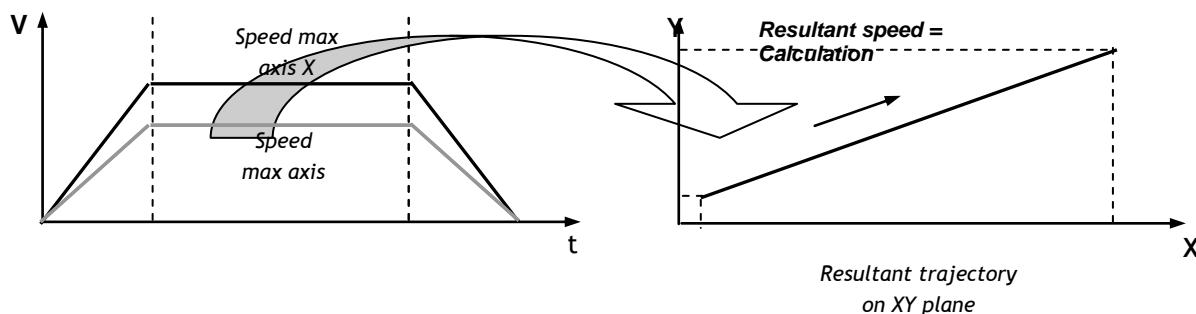
The axes defined in GMC_MoveUntilVariable start and end their movements simultaneously; this means that they are “interpolated” with one another and hence the trajectory specified for the movement will be maintained. From the dynamic standpoint, each axis adopts a speed curve of its own, regardless of the other axes in terms of feed rate reached and acceleration and jerk values; since the motions of the various axes are mutually interpolated, the acceleration stage ends at the same time for all the axes: similarly, the deceleration stage starts simultaneously for all axes.



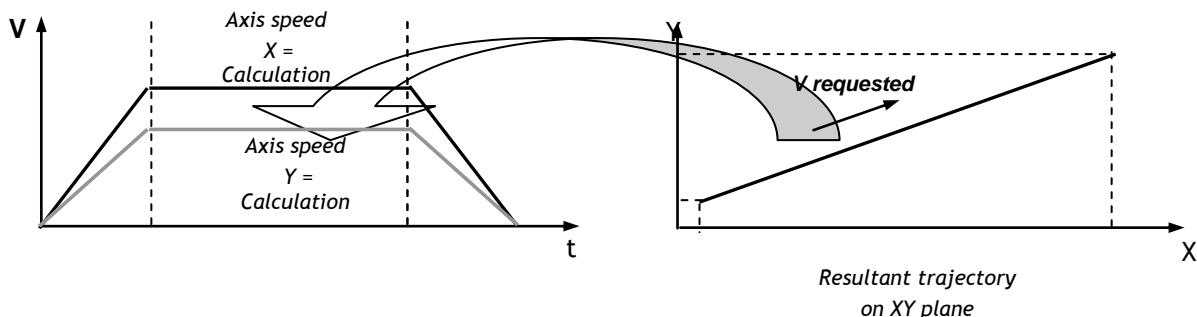
The system makes sure that the dynamic parameters requested for each axis do not exceed the (feed, acceleration and jerk) values programmed for it. If the dynamic values requested for an axis are exceedingly high, the system will limit the dynamics of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes having different clocks must be moved separately.

Programmed feed

As a rule, movements are performed at the highest possible feed rate, which is determined as a function of the maximum rate that each individual axis can reach. The speed obtained over a trajectory therefore depends both on the maximum feed that an axis can reach and on the extent to which an axis moves relative to global motion (ratio between the distance covered by the axis and the distance covered by the global movement).



Moreover, it is possible to set the “speed on the profile”, i.e., resultant speed on the global movement; in this case the speed at which an axis moves will be determined starting from the speed on the profile and the component of the movement of the individual axis relative to the overall distance to be covered “on the profile”.



Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated

0x0016000D	Interpolation activities exhausted
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160037	VelMode parameter wrong
0x00160034	Movement stop condition wrong
0x00160005	Too many axes specified
0x00160048	StopVar synchronism variable wrong
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(*Executes movement of axis with ID 1 at point 100.0, stopping if
 variable MI33 takes on value of 17 indicating stop on M0_0 *)
Ret := GMC_MoveUntil Variable (MODE_WAIT, UL0, InterpId, M0_0, MI33,
                               MODE_FAST, 0.0, 17, CND_RAISE, Feed, 1, 100.0);

```

See also:

[GMC_SetRamp](#), [GMC_SetJerk](#), [GMC_SetFeedOverride](#), [GMC_AbsoluteQuote](#), [GMC_IncrementalQuote](#), [GMC_ActivateOrigin](#)

3.24 GMC_ToleranceParams

Function **GMC_ToleranceParams** defines the tolerance for the calculation of circular trajectories.

Syntax:

```
ret_code := GMC_ToleranceParams (ExecMode, ExecStatus, InterId, Precision,
RadErr, FullCirc, RadMode) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	Interpolator identifier PLC [1..40]
<i>Precision</i> (LREAL)	Geometric precision
<i>RadErr</i> (LREAL)	Error in determination of centre of circumference (CET)
<i>FullCirc</i> (LREAL)	Full circle threshold (FCT)
<i>RadMode</i> (INT)	Circumference recalculation mode (ARM)

Execution Mode:

Wait / NoWait

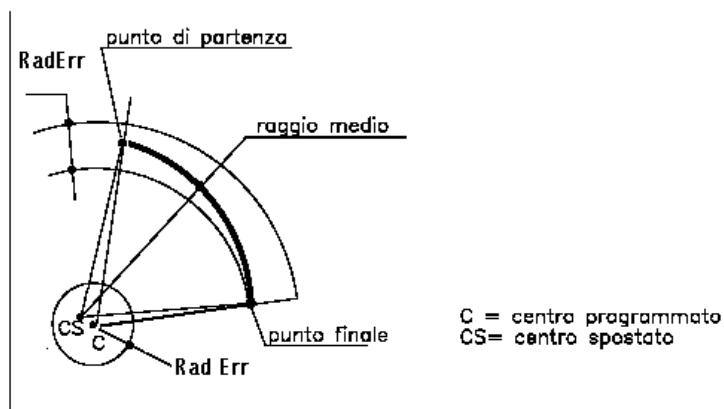
Use:

This function is used to configure the parameters affecting the calculation of circular motions. In particular, this applies to the configuration of the following variables: **RadErr**, which defines the maximum error admissible in defining the points associated with circular motions (difference between the values of the radii subtended between centre/end point and centre/start point), **FullCirc** (tolerance to decide whether to execute a full circle or a small arc of a circle), and **RadMode**, which defines the criterion for the automatic recalculation of the coordinates of the centre of the circumference when an incorrect one has been programmed. The **RadMode** parameter may apply. The geometric tolerance threshold **Precision** is also defined.

The **RadMode** parameter is defined according to the following methodology:

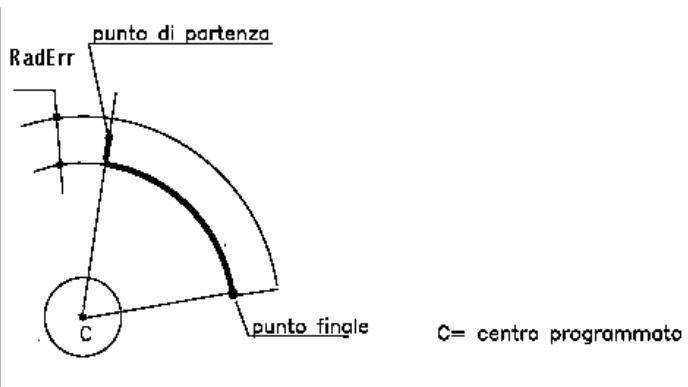
RadMode =	0 MOD_CENTOL	Recalculation of the centre within the RadErr tolerance
	1 MOD_PINI	Moves the initial point to within the RadErr tolerance
	2 MOD_CEN	Recalculation of the centre in an arbitrary manner
	3 MOD_POINTS	Move the starting point and the end point to within the RadErr tolerance

RadMode =0 Recalculation of the centre of a circumference to within a certain tolerance



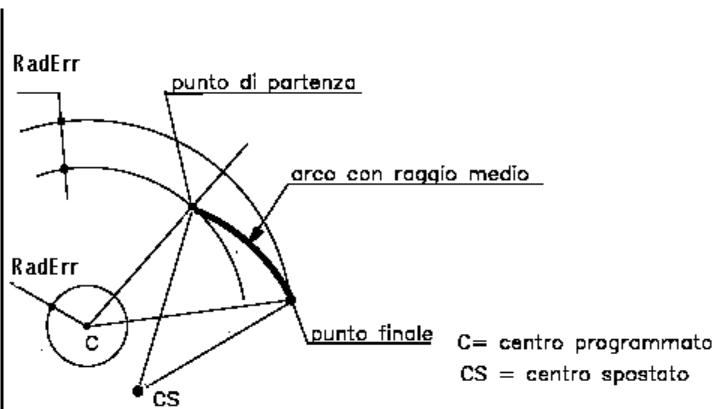
A circumference passing through the starting and end points is executed having its centre within a tolerance defined by parameter RadErr. In this case the circumference is executed with a mean radius.

RadMode =1 Moving the starting point to within a certain tolerance



The circumference executed passes through a starting point corrected to fall within the tolerance defined with RadErr and an end point as programmed. The centre remains unchanged.

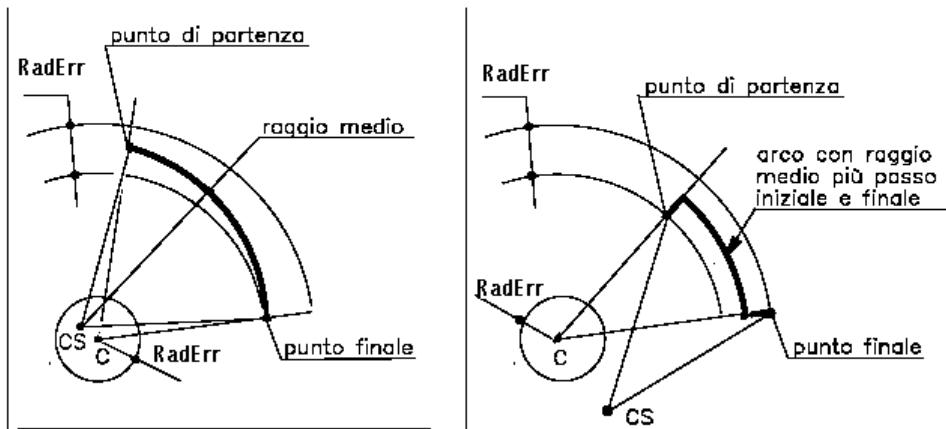
RadMode =2 Recalculation of the centre of the circumference



The circumference passes through the starting point and the end point and its centre is moved without considering the tolerance. In this case the circumference is executed with a mean radius.

RadMode =3 The centre is maintained and the starting and end points are moved

C = centro programmato
CS = centro spostato



If the movement of the centre remains within the tolerance range defined by RadErr, then the circumference executed passes through the starting and end points and has its centre moved to a new position (same as with RadMode=0). If the movement is not within the tolerance specified by RadErr, then the centre remains where it was and the starting and end points are corrected by a value corresponding to RadErr/2.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0016005C	RadMode parameter wrong
0x00160014	Interpolator not present

Example:

```
Ret      : dword;
InterpId : INT;
...
Ret := GMC_ToleranceParams (MODE_WAIT, UI0, InterpId, 0.01, 0.05,
                           0.05, MOD_CEN) ;
```

See also:

GMC_CircleCW, GMC_CircleCCW, GMC_CircleCWRadius, GMC_CircleCCWRadius,

3.25 GMC_CircleCCW

Function GMC_CircleCCW executes a counter clockwise circular movement with a given centre.

Syntax:

```
ret_code := GMC_CircleCCW (ExecMode, ExecStatus.InterId, EndSignal, Feed,  
                  CenterX, CenterY, AxisId, Position) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	Interpolator identifier PLC [1..40]
<i>Feed</i> (LREAL)	Feed in units/min
<i>CenterX</i>	X axis centre coordinate
<i>CenterY</i>	Y axis centre coordinate
<i>AxisId</i> (INT)	Axis identifier [1...64] for max 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters:

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (a TRUE)
-------------------------	--

Possible overload:

GMC_CircleCCW (INT,DWORD,INT,BOOL, FeedDescr_Struct, LREAL,LREAL,
 INT,LREAL,INT,LREAL[,INT,LREAL]...)

Execution mode:

Wait / NoWait

Use:

This function makes it possible to execute a counter clockwise movement, which is either circular (if only two axes have been programmed) or helicoidal (more than 2 axes programmed), whose centre is located at a given point. The two axes that are programmed first are taken to be the abscissa and ordinate axes. At the end of the execution of the movement, signal “*EndSignal*” is set to true. Parameters “*AxisId*” and “*Position*” may be duplicated in order to request the synchronous movement of several axes, up to a maximum of 12 (2 as a minimum).

The movement is executed at fast speed or at a speed value defined by the “*Feed*” parameter. This structure allows to program the speed, acceleration, deceleration and jerk values which will be used during movement. If these fields are not set, the movement will be executed by default dynamics.

The following table describes the fields of the *Feed* structure.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Here follows the values that can be adopted by the *TypeFeed* variable.

TypeFeed = 0: MODE_RAPID	Movements will use the rapid speed rate and the dynamic parameters of the rapid or programmed movements.
1: MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2: MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3: MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

The movement is executed at the speed defined by the “*Feed*” parameter, expressed as unit/minute. Speed is to be conceived as vector speed, i.e., the rate of displacement of the set of axes moved, not that of the individual axes. Execution speed is limited based on the size of the radius to be executed, so as to limit the centrifugal acceleration that is generated by the circular motion. The limit is imposed by considering the smaller between X or Y axis acceleration.

$$\text{Feed} < \sqrt{A * \text{Radius}} \quad A = \min(\text{accX}, \text{accY})$$

The “*Position*” parameter may take on different meanings as a function of the axis programming context. In the case of absolute programming, this parameter refers to the point the axis will be at (end point) at end of movement, in the case of incremental programming it defines the distance (preceded by a sign) that the axis must cover starting from its current position. All end points are taken to refer to the origin activated for the programmed axis. If the operating limits are active, they are verified before executing the movement, and this check may generate an Over travel error.

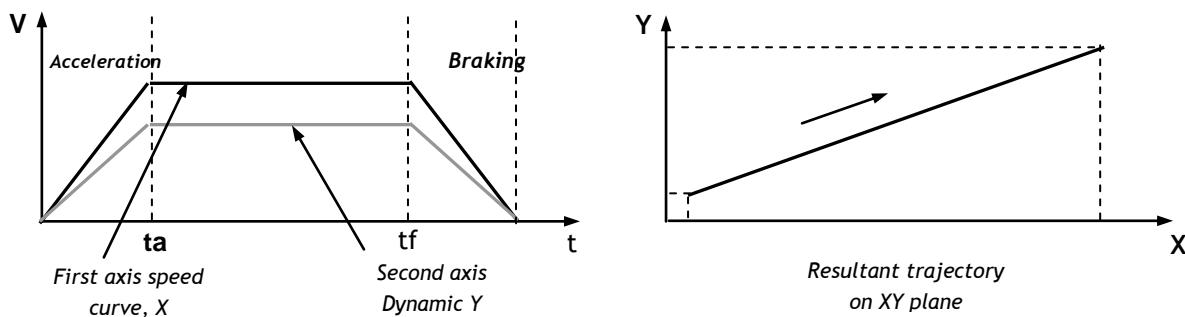
Even for parameters “*CenterX*” and “*CenterY*” absolute/incremental programming mode and the origins are taken into account; keep in mind that “*CenterX*” is associated with the axis that has been programmed for first and “*CenterY*” is associated with the second.



This function may be programmed from within the continuous stage

Coordinating the axes of a movement

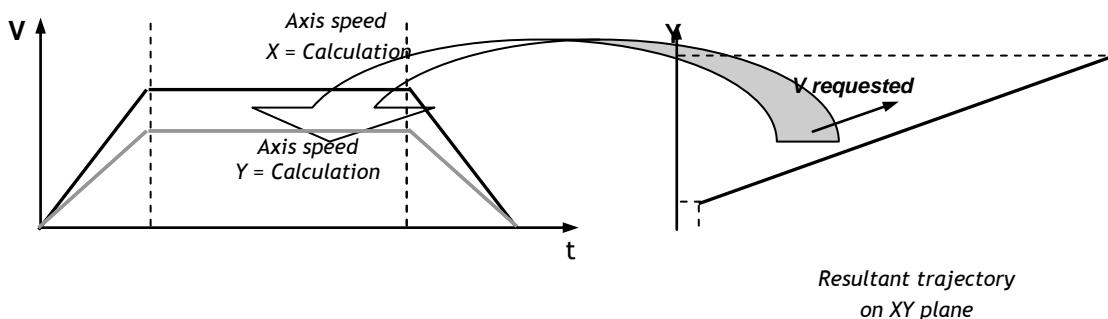
The axes defined in GMC_CircleCCW *start and end their movements simultaneously*; hence they are mutually “interpolated” and therefore the required trajectory is maintained. From the dynamic standpoint each axis will adopt a speed curve of its own, independent of the other axes in terms of feed rates reached and accelerations and jerks adopted; since the motions of the various axes are mutually interpolated, the acceleration stage ends simultaneously for all the axes; similarly, the deceleration stage begins simultaneously.



In any event, checks are carried out to determine whether the speed curves requested for each axis exceed the (feed, acceleration and jerk) values configured for the axis in question. If the speed curves requested for an axis are too high, limits are set on the speed curves of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes with different clocks will have to be moved separately.

Programmed speed

This function works according to the principle of “speed on the profile”, i.e., the speed obtained for the movement as a whole; by this principle, the speed at which an axis moves is calculated based on speed on the profile and the motion component of the individual axis relative to the overall distance to be covered “on the profile”.



Circular motions

The programming of circular motions (also referred to as circumferences) is susceptible to programming inaccuracies (numerical representation) and hence the circles programmed are often rejected in as much as they do NOT describe a perfect circle. To remedy this problem, “tolerances” and “methodologies” are provided for use in defining the circles. In this connection, see function GMC_ToleranceParams.

Full circles

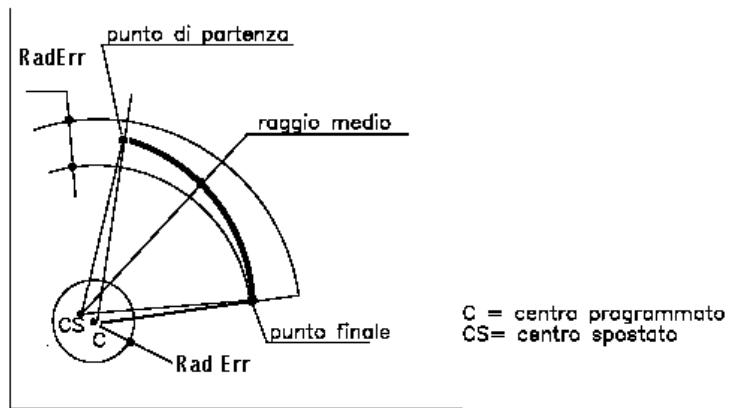
In programming a full circle, the end point must be made to coincide with the starting point. There is a system variable, **FullCirc** (FCT), by which if the distance between the starting point and the end point is smaller than this constant, the circle executed will be a full circumference. For example, assuming our circle has a starting point (100, 100) and an end point (100.1, 100.1), it will be executed as an arc if FullCirc=0.1, and will be executed as a full circle (with end point 100,100) if FullCirc=0.15, the distance between the two points being 0.14142.

Circle execution modes

Another problem to be addressed in connection with the programming of circular motions is the precision required in programming the centre in order to describe a perfect circumference. First of all, in programming a circumference, the radii subtended between the start and end point are determined: if the differences between the radii is nil, the circle is executed as requested, if the difference exceeds a given threshold, as defined by system variable **RadErr (CET)**, an error is generated, if the difference is in the 0 - RadErr range, methods are provided for recalculating the circumference parameters so that the circumference may be defined in a congruent manner. Four different programming modes, defined by means of system variable **RadMode (ARM)**, are available, to:

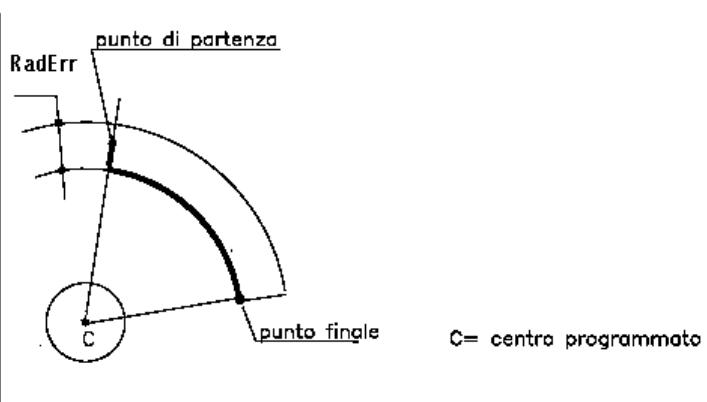
- ▷ Recalculate the centre of a circle within the tolerance defined by RadErr
- ▷ Move the starting point to within the tolerance defined by RadErr
- ▷ Recalculate the centre of a circle irrespective of RadErr (default mode)
- ▷ Maintain the centre where it is and move the starting and end points to within the tolerance defined by RadErr

RadMode = 0 Recalculating the centre of a circle within a certain tolerance

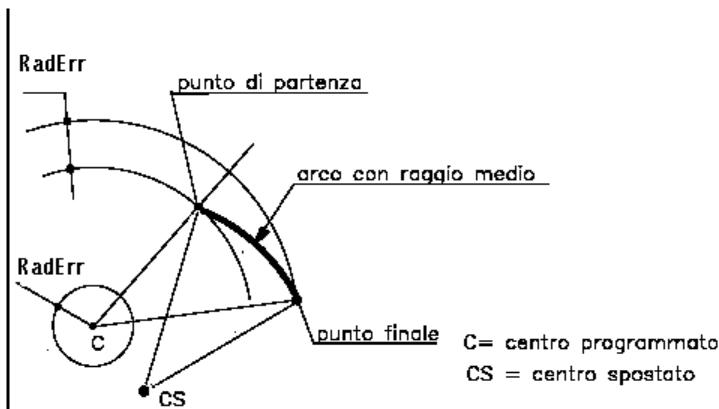
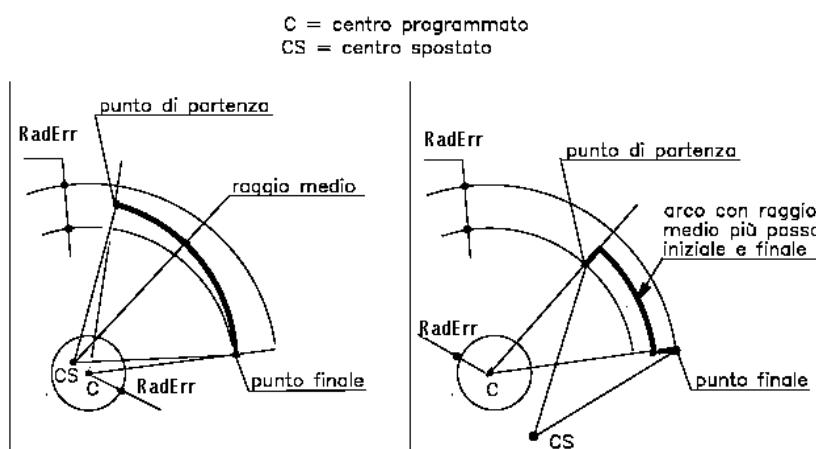


The circumference is recalculated so that it passes through the starting and end points and its centre is moved to within the tolerance range defined by parameter RadErr. In this case the circle is executed with a mean radius.

RadMode = 1 Starting point moved to within a certain tolerance



The circumference is recalculated so that it passes through a starting point moved to within the tolerance range defined by parameter RadErr and through the end point programmed. The centre is not moved.

RadMode =2 Recalculating the centre of a circle**RadMode =3 Keeping the centre where it is and moving the starting and end points****Return values:**

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160003	Too many activities in progress
0x00160014	Interpolator not present
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle

0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities have been used up
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(* Executes a counter clockwise circular movement at a feed rate of 10000
mm/min at point 100 for the first axis and point 0 for the second,
the centre is at 100.0, 100.0; a 90° circle is executed.

At end of movement variable M0_0 is set to true *) 
Feed.Value    := 10000.0 ;
Feed.TypeFeed := MODE_WORK;
Ret := GMC_CircleCCW (MODE_WAIT, ULO, InterpId, M0_0, Feed,
                      100.0, 100.0, 1, 100.0, 2, 0.0) ;

```

See also :

GMC_SetRamp, GMC_SetJerk, GMC_SetFeedOverride, GMC_AbsoluteQuote, GMC_IncrementalQuote, GMC_ActivateOrigin, GMC_ToleranceParms

3.26 GMC_CircleCCWRadius

Function GMC_CircleCCWRadius executes a counter clockwise circular movement with a given centre.

Syntax:

```
ret_code := GMC_CircleCCWRadius (ExecMode, ExecStatus, Interpld, EndSignal, Feed,
Radius, AxisId, Position) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>Feed</i> (LREAL)	Speed in units/min
<i>Radius</i>	Radius with sign
<i>AxisId</i> (INT)	Axis identifier [1...64] for max 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters:

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (a TRUE)
-------------------------	--

Possible overload:

GMC_CircleCCWRadius	(INT, DWORD, INT, BOOL, FeedDescr_Struct, LREAL, INT, LREAL, INT, LREAL[, INT, LREAL]...)
---------------------	--

Execution mode:

Wait / NoWait

Use:

This function makes it possible to execute a counter clockwise movement, which is either circular (if only two axes have been programmed) or helicoidal (more than 2 axes programmed), whose centre is located at a given point. The two axes that are programmed first are taken to be the abscissa and ordinate axes. At the end of the execution of the movement, signal “*EndSignal*” is set to true. Parameters “*AxisId*” and “*Position*” may be duplicated in order to request the synchronous movement of several axes, up to a maximum of 12 (2 as a minimum).

The movement is executed at fast speed or at a speed value defined by the “*Feed*” parameter. This structure defines the speed, acceleration, deceleration and jerk values which will be used during movement. If these fields are not set, the movement will be executed by default dynamics.

The following table describes the fields of the *Feed* structure.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Here follows the values that can be adopted by the *TypeFeed* variable:

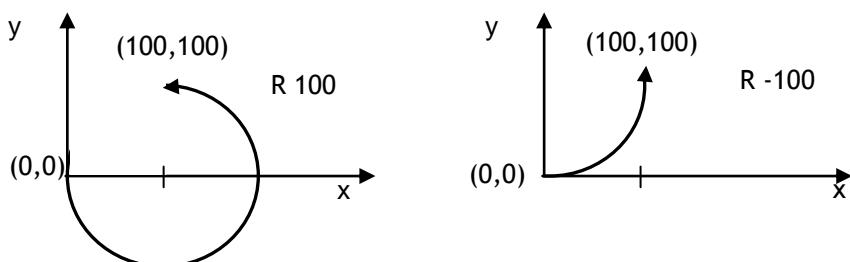
- | | |
|--------------------------|---|
| TypeFeed = 0: MODE_RAPID | Movements will use the rapid speed rate and the dynamic parameters of the rapid or programmed movements. |
| 1: MODE_WORK | Movements will use the programmed speed and the dynamic parameters of programmed or work movement |
| 2: MODE_FEED_t | Movements are programmed in time and will use the dynamic parameters of work or programmed movements. |
| 3: MODE_FEED_1T | Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements. |

The movement is executed at the speed defined by the “*Feed*” parameter, expressed as unit/minute. Speed is to be conceived as vector speed, i.e., the rate of displacement of the set of axes moved, not that of the individual axes. Execution speed is limited based on the size of the radius to be executed, so as to limit the centrifugal acceleration that is generated by the circular motion. The limit is imposed by considering the acceleration of either the X or the Y axis, whichever is smaller.

$$\text{Feed} < \sqrt{A * \text{Radius}} \quad A = \min(\text{accX}, \text{accY})$$

The “*Position*” parameter may take on different meanings as a function of the axis programming context. In the case of absolute programming, this parameter refers to the point the axis will be at (end point) at end of movement, in the case of incremental programming it defines the distance (preceded by a sign) that the axis must cover starting from its current position. All end points are taken to refer to the origin activated for the programmed axis. If the operating limits are active, they are verified before executing the movement, and this check may generate an Over travel error.

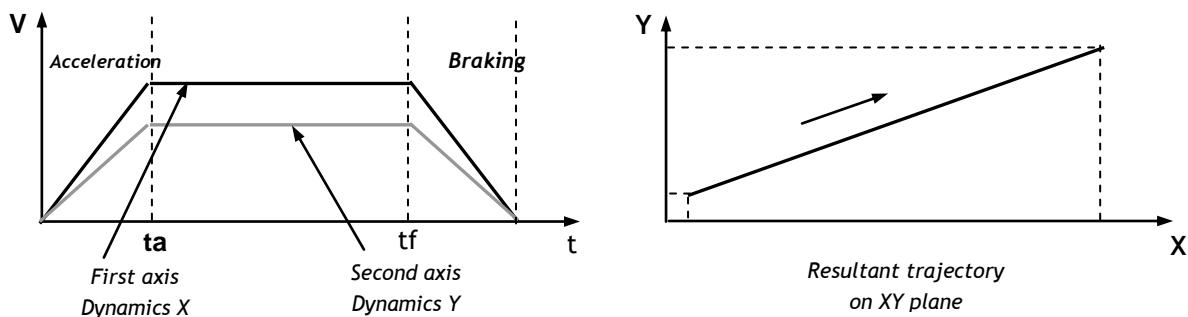
The “*Radius*” parameter has a sign which determines which of the two possible circles is going to be executed; if the sign is positive, the longer circle is executed, if it is negative, the shorter circle is selected.



This function may be programmed from within the continuous stage

Coordinating the axes of a movement

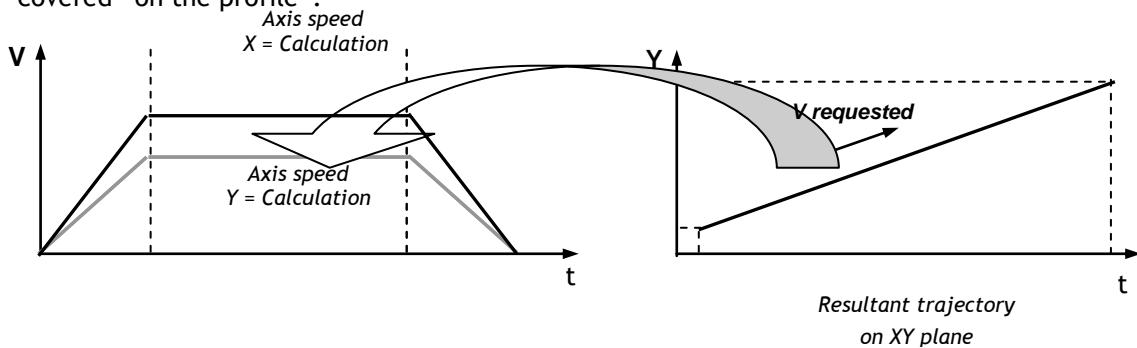
The axes defined in GMC_CircleCCWRRadius start and end their movements simultaneously; hence they are mutually “interpolated” and therefore the required trajectory is maintained. From the dynamic standpoint each axis will adopt a speed curve of its own, independent of the other axes in terms of feed rates reached and accelerations and jerks adopted; since the motions of the various axes are mutually interpolated, the acceleration stage ends simultaneously for all the axes; similarly, the deceleration stage begins simultaneously.



In any event, checks are carried out to determine whether the speed curves requested for each axis exceed the (feed, acceleration and jerk) values configured for the axis in question. If the speed curves requested for an axis are too high, limits are set on the speed curves of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes with different clocks will have to be moved separately.

Programmed speed

This function works according to the principle of “speed on the profile”, i.e., the speed obtained for the movement as a whole; by this principle, the speed at which an axis moves is calculated based on speed on the profile and the motion component of the individual axis relative to the overall distance to be covered “on the profile”.



Circular motions

The programming of circular motions (also referred to as circumferences) is susceptible to programming inaccuracies (numerical representation) and hence the circles programmed are often rejected in as much as they do NOT describe a perfect circle. To remedy this problem, “tolerances” and “methods” are provided for defining the circles. In this connection, see function GMC_ToleranceParams.

Full circles

In programming a full circle, the end point must be made to coincide with the starting point. There is a system variable, **FullCirc (FCT)**, by which if the distance between the starting point and the end point is smaller than this constant, the circle executed will be a full circumference. For example, assuming our circle has a starting point (100, 100) and an end point (100.1,100.1), it will be executed as an arc if **FullCirc=0.1**, and will be executed as a full circle (with end point 100,100) if **FullCirc=0.15**, the distance between the two points being 0.14142.

Circle execution modes

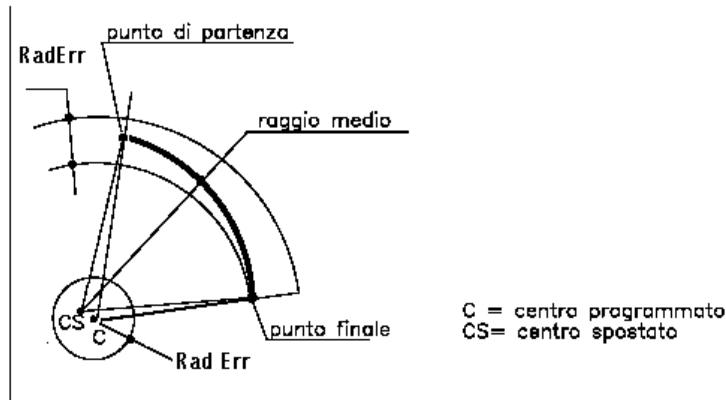
Another problem to be addressed in connection with the programming of circular motions is the precision required in programming the centre in order to describe a perfect circumference.

First of all, in programming a circumference, determine the radii subtended between the start and end point: if the differences between the radii is nil, the circle is executed as requested, if the difference exceeds a given threshold, as defined by system variable **RadErr (CET)**, an error is generated, if the difference is in the 0 - RadErr range, methods are provided for recalculating the circumference parameters so that the circumference may be defined in a congruent manner.

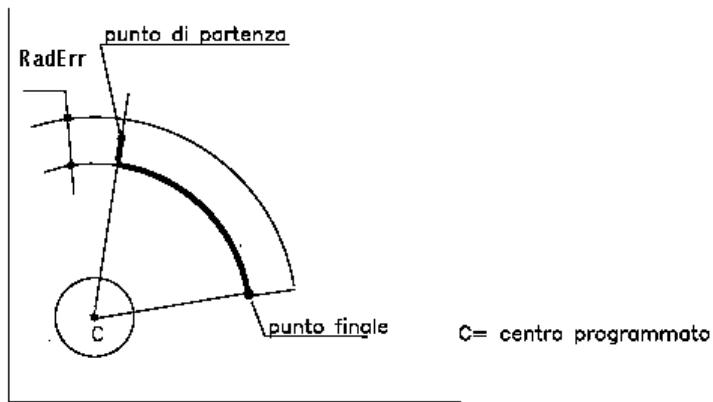
Four different programming modes, defined by means of system variable **RadMode (ARM)**, are available, to:

- ▷ Recalculate the centre of a circle within the tolerance defined by RadErr
- ▷ Move the starting point to within the tolerance defined by RadErr
- ▷ Recalculate the centre of a circle irrespective of RadErr (default mode)
- ▷ Maintain the centre where it is and move the starting and end points to within the tolerance defined by RadErr

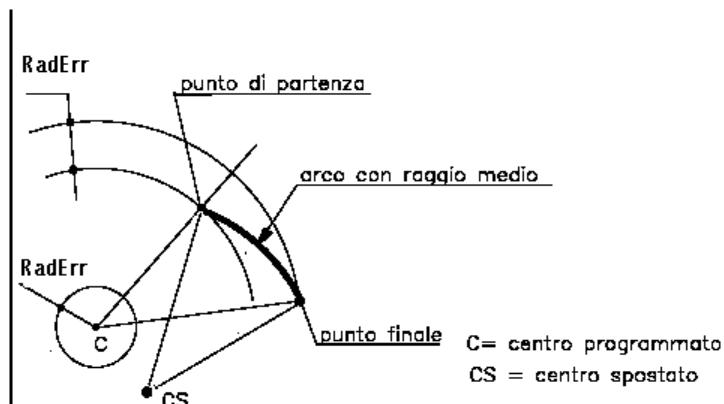
RadMode =0 Recalculating the centre of a circle within a certain tolerance



The circumference is recalculated so that it passes through the starting and end points and its centre is moved to within the tolerance range defined by parameter RadErr. In this case the circle is executed with a mean radius.

RadMode =1 Starting point moved to within a certain tolerance

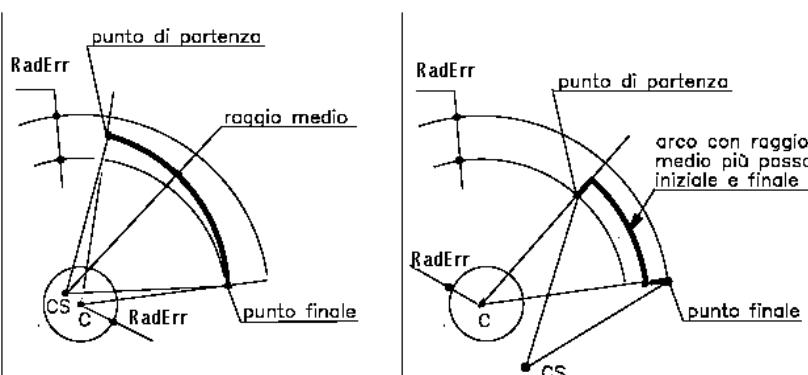
The circumference is recalculated so that it passes through a starting point moved to within the tolerance range defined by parameter $RadErr$ and through the end point programmed. The centre is not moved.

RadMode =2 Recalculating the centre of a circle

The circumference is recalculated so that it passes through the starting and end points and its centre is moved regardless of tolerance constraints. In this case the circle is executed with a mean radius.

RadMode =3 Keeping the centre where it is and moving the starting and end points

C = centro programmato
 CS = centro spostato



If the movement of the centre of the circle takes place within the tolerance range defined by $RadErr$, then the circumference is recalculated so that it passes through the starting and end points and its centre is moved appropriately. (same as in $RadMode=0$). If the movement of the circle is not within the tolerance specified by $RadErr$, then the centre is maintained where it is and the starting and end points are corrected by a value corresponding to $RadErr/2$.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities have been used up
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(*Executes a counter clockwise circular movement at a feed rate of 10000
mm/min at point 100 for the first axis and point 100 for the second,
the centre is at 100.0, a 270° circle is executed.

At end of movement variable M0_0 is set to true *)
Feed.Value    := 10000.0 ;
Feed.TypeFeed := MODE_WORK;
Ret := GMC_CircleCCWRadius (MODE_WAIT, UL0, InterpId, M0_0, Feed,
                           100.0, 1, 100.0, 2, 100.0) ;

```

See also:

GMC_SetRamp, GMC_SetJerk, GMC_SetFeedOverride, GMC_AbsoluteQuote, GMC_IncrementalQuote, GMC_ActivateOrigin, GMC_ToleranceParms

3.27 GMC_CircleCW

Function **GMC_CircleCW** executes a clockwise circular movement with a given centre.

Syntax:

```
ret_code := GMC_CircleCW (ExecMode, ExecStatus, Interpld, EndSignal, Feed,  
                  CenterX, CenterY, AxisId, Position) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>Feed</i> (LREAL)	Speed in units/min
<i>CenterX</i>	Abscissa axis centre coordinate
<i>CenterY</i>	Ordinate axis centre coordinate
<i>AxisId</i> (INT)	Axis identifier [1...64] for max 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (a TRUE)
-------------------------	--

Possible overload:

GMC_CircleCW (INT, DWORD, INT, BOOL, FeedDescr_Struct, LREAL, LREAL,
 INT, LREAL, INT, LREAL[, INT, LREAL]...)

Execution Mode:

Wait / NoWait

Use:

This function makes it possible to execute a clockwise movement, which is either circular (if only two axes have been programmed) or helicoidal (more than 2 axes programmed), whose centre is located at a given point. The two axes that are programmed first are taken to be the abscissa and ordinate axes. At the end of the execution of the movement, signal “*EndSignal*” is set to true. Parameters “*AxisId*” and “*Position*” may be duplicated in order to request the synchronous movement of several axes, up to a maximum of 12 (2 as a minimum).

The movement is executed at fast speed or at a speed value defined by the “*Feed*” parameter. This structure defines the speed, acceleration, deceleration and jerk values which will be used during movement. If these fields are not set, the movement will be executed by default dynamics.

The following table describes the fields of the *Feed* structure.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Starting from OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration Jerk (unit/s ³)
TminAcc	LREAL	Acceleration minimum ramp time (ms)
TminDec	LREAL	Deceleration minimum ramp time (ms)

Here follows the values that can be adopted by the *TypeFeed* variable.

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed rate and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

The movement is executed at the speed defined by the “*Feed*” parameter, expressed as unit/minute. Speed is to be conceived as vector speed, i.e., the rate of displacement of the set of axes moved, not that of the individual axes. Execution speed is limited based on the size of the radius to be executed, so as to limit the centrifugal acceleration that is generated by the circular motion. The limit is imposed by considering the smaller acceleration value between X or Y axis.

$$\text{Feed} < \sqrt{A * \text{Radius}} \quad A = \min(\text{accX}, \text{accY})$$

The “*Position*” parameter may take on different meanings as a function of the axis programming context. In the case of absolute programming, this parameter refers to the point the axis will be at (end point) at end of movement, in the case of incremental programming it defines the distance (preceded by a sign) that the axis must cover starting from its current position. All end points are taken to refer to the origin activated for the programmed axis. If the operating limits are active, they are verified before executing the movement, and this check may generate an Over travel error.

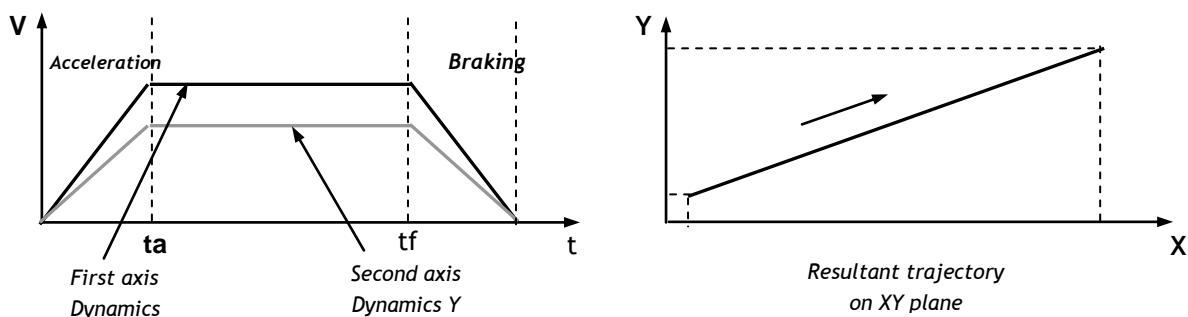
Even for parameters “CenterX” and “CenterY” the absolute/incremental programming mode and the origins are considered; keep in mind that “CenterX” is associated with the first axis programmed and “CenterY” is associated with the second.



This function may be programmed from within the continuous stage

Coordinating the axes of a movement

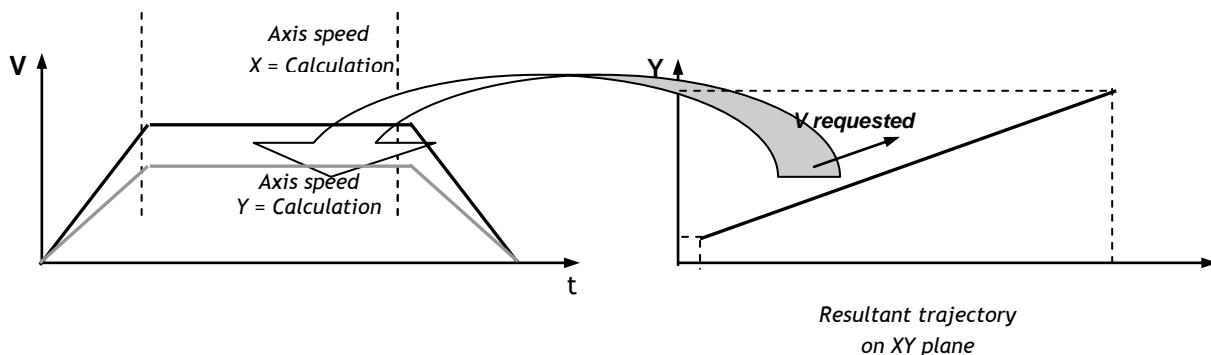
The axes defined in GMC_CircleCW start and end their movements simultaneously; hence they are mutually “interpolated” and therefore the required trajectory is maintained. From the dynamic standpoint each axis will adopt a speed curve of its own, independent of the other axes in terms of feed rates reached and accelerations and jerks adopted; since the motions of the various axes are mutually interpolated, the acceleration stage ends simultaneously for all the axes; similarly, the deceleration stage begins simultaneously.



In any event, checks are carried out to determine whether the speed curves requested for each axis exceed the (feed, acceleration and jerk) values configured for the axis in question. If the speed curves requested for an axis are too high, limits are set on the speed curves of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes with different clocks will have to be moved separately.

Programmed speed

This function works according to the principle of “speed on the profile”, i.e., the speed obtained for the movement as a whole; by this principle, the speed at which an axis moves is calculated based on speed on the profile and the motion component of the individual axis relative to the overall distance to be covered “on the profile”.



Circular motions

The programming of circular motions (also referred to as circumferences) is susceptible to programming inaccuracies (numerical representation) and hence the circles programmed are often rejected in as much as they do NOT describe a perfect circle. To remedy this problem, "tolerances" and "methodologies" are provided for use in defining the circles. In this connection, see function GMC_ToleranceParam.

Full circles

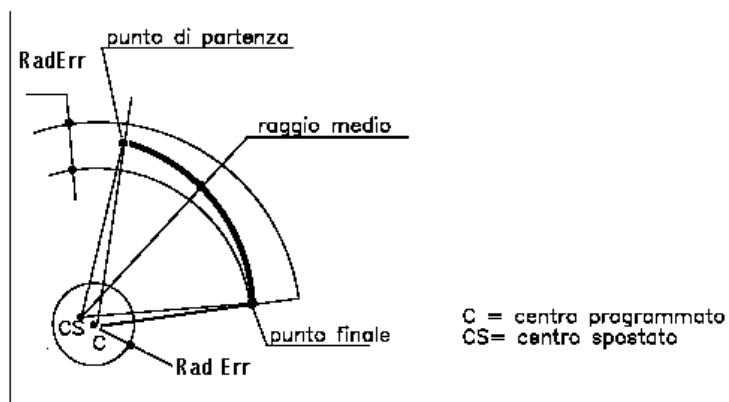
In programming a full circle, the end point must be made to coincide with the starting point. There is a system variable, **FullCirc (FCT)**, by which if the distance between the starting point and the end point is smaller than this constant, the circle executed will be a full circumference. For example, assuming our circle has a starting point (100, 100) and an end point (100.1,100.1), it will be executed as an arc if FullCirc=0.1, and will be executed as a full circle (with end point 100,100) if FullCirc=0.15, the distance between the two points being 0.14142.

Circle execution modalities

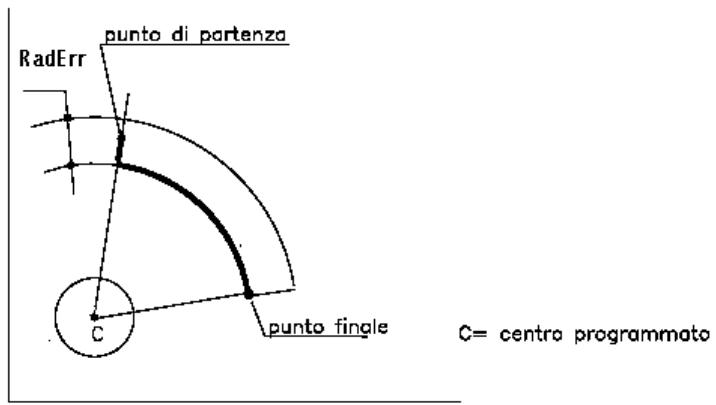
Another problem to be addressed in connection with the programming of circular motions is the precision required in programming the centre in order to describe a perfect circumference. First of all, in programming a circumference, the radii subtended between the start and end point are determined: if the differences between the radii is nil, the circle is executed as requested, if the difference exceeds a given threshold, as defined by system variable **RadErr (CET)**, an error is generated, if the difference is in the 0 - RadErr range, methodologies are provided for recalculating the circumference parameters so that the circumference may be defined in a congruent manner. Four different programming modes, defined by means of system variable **RadMode (ARM)**, are available, to:

- ▷ Recalculate the centre of a circle within the tolerance defined by RadErr
- ▷ Move the starting point to within the tolerance defined by RadErr
- ▷ Recalculate the centre of a circle irrespective of RadErr (default mode)
- ▷ Maintain the centre where it is and move the starting and end points to within the tolerance defined by RadErr

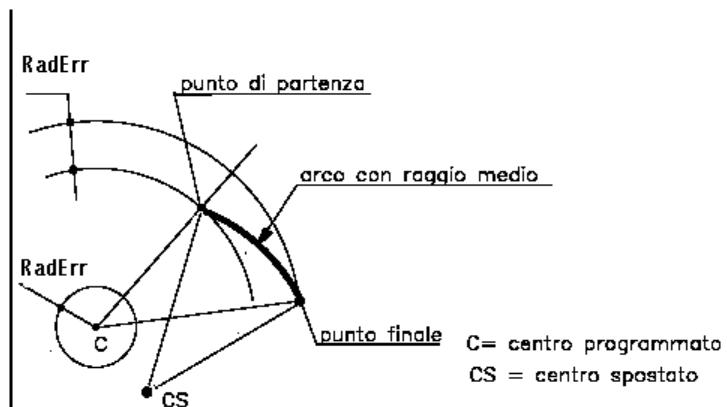
RadMode =0 Recalculating the centre of a circle within a certain tolerance



The circumference is recalculated so that it passes through the starting and end points and its centre is moved to within the tolerance range defined by parameter RadErr. In this case the circle is executed with a mean radius.

RadMode =1 Starting point moved to within a certain tolerance

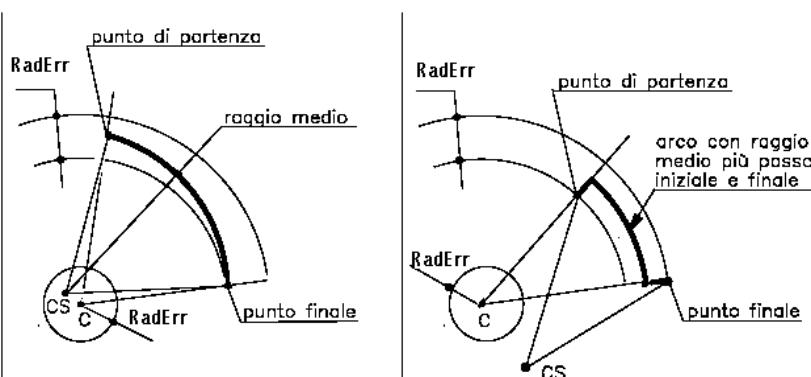
The circumference is recalculated so that it passes through a starting point moved to within the tolerance range defined by parameter RadErr and through the end point programmed. The centre is not moved.

RadMode =2 Recalculating the centre of a circle

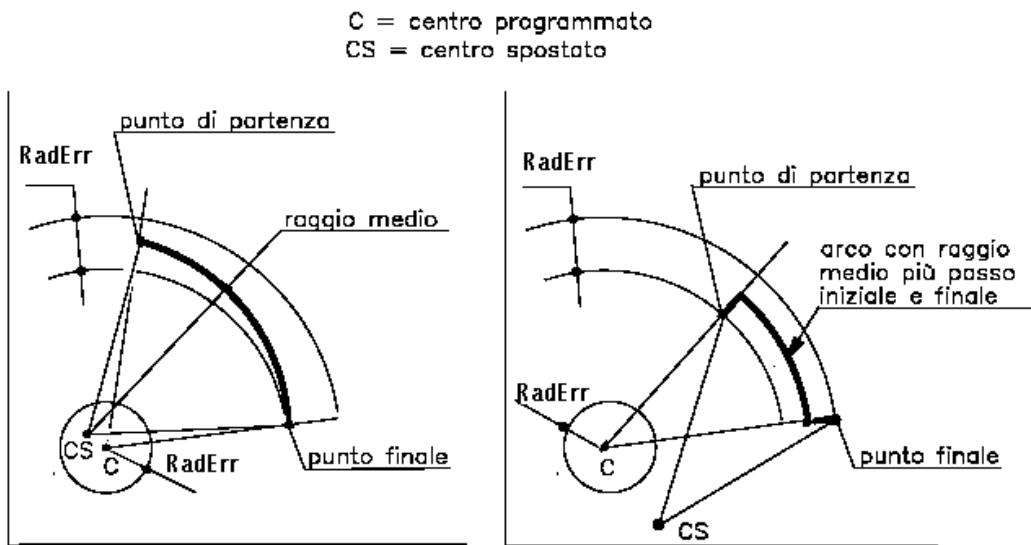
The circumference is recalculated so that it passes through the starting and end points and its centre is moved regardless of tolerance constraints. In this case the circle is executed with a mean radius.

RadMode =3 Keeping the centre where it is and moving the starting and end points

C = centro programmato
 CS = centro spostato



If the movement of the centre of the circle takes place within the tolerance range defined by RadErr , then the circumference is recalculated so that it passes through the starting and end points and its centre is moved appropriately. (same as in $\text{RadMode}=0$). If the movement of the circle is not within the tolerance specified by RadErr , then the centre is maintained where it is and the starting and end points are corrected by a value corresponding to $\text{RadErr}/2$.

**Return values:**

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities have been used up
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```
Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(*Executes a clockwise circular movement at a feed rate of 10000
mm/min at point 100 for the first axis and point -100 for the second,
the centre is at 100.0, 0.0; a 270° circle is executed.

At end of movement variable M0_0 is set to true*)

Feed.Value    := 10000.0 ;
Feed.TypeFeed := MODE_WORK;
Ret := GMC_CircleCW (MODE_WAIT, ULO, InterpId, M0_0, Feed,
                     100.0, 0.0, 1, 100.0, 2, -100.0) ;
```

See also :

GMC_SetRamp, GMC_SetJerk, GMC_SetFeedOverride, GMC_AbsoluteQuote, GMC_IncrementalQuote,
GMC_ActivateOrigin, GMC_ToleranceParms

3.28 GMC_CreateAxVirtual

GMC_CreateAxVirtual function inserts the virtual axis within the axes managed by the interpolator.

Syntax

```
ret_code := GMC_CreateAxVirtual (AxisId, Name) ;
```

Input parameters

<i>Id</i> (INT)	Axis id [1...64]
<i>Name</i> (STRING)	Axis name

Execution mode

Immediate

Use

The function allows the interpolator to manage a virtual axis previously created with RS_CreateResources function. *Id* is the identifier returned by the RS_CreateResources function. *Name* is the name to be associated with the axis.

Before the motion of the axis created, it is necessary to configure at least the parameters related to the dynamic by AX_WriteParameter function. It's very important to set the following parameters.

- fn_RAPID_TRAVERSE_FEED
- fn_RAPID_ACCELERATION
- fn_MANUAL_FEED
- fn_MANUAL_ACCELERATION
- fn_RAPID_JERK
- fn_WORKING_JERK
- fn_RAPID_TIME_MIN_RAMP
- fn_MANUAL_TIME_MIN_RAMP
- fn_RAPID_DECELERATION
- fn_MANUAL_DECELERATION
- fn_RAPID_DJERK
- fn_WORKING_DJERK
- fn_RAPID_TIME_MIN_DRAMP
- fn_WORKING_TIME_MIN_DRAMP

Return values

The following table resumes the possible values *ret_code* variable

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x001600DB	Id does not correspond to a virtual axis
0x00200004	Wrong Id
0x00200006	Axis id not present

Example

```
ret      : dword;
Name     : STRING := 'W';
Id       : INT  := 1;

(*Virtual axis with Id = 1 and Name = 'W' association *)
ret := GMC_CreateAxVirtual (Id, Name);
```

See also

[AX_WriteParameter](#), [GMC_DeleteAxVirtual](#), [RS_CreateResource](#), [RS_DestroyResource](#)

3.29 GMC_DeleteAxVirtual

GMC_DeleteAxVirtual function deletes the virtual axis from the table showing the axes managed by the interpolator.

Syntax

```
ret_code := GMC_DeleteAxVirtual (Id) ;
```

Input parameters

<i>Id</i> (INT)	Axis id [1...64]
-----------------	------------------

Execution mode

Immediate

Use

The function does not allow the interpolator to keep on managing in input the virtual axis.

Return values

The following table shows the values the *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x001600DB	Id does not correspond to a virtual axis
0x00200004	Wrong Id
0x00200006	Axis id not present

Example

```
ret      : dword;
Name     : STRING := 'W';
Id       : INT  := 1;

(* Remove from the interpolator the virtual axis with Id = 1 *)
ret := GMC_DeleteAxVirtual(Id, Name);
```

See also

[GMC_CreateAxVirtual](#), [RS_CreateResource](#), [RS_DestroyResource](#)

3.30 GMC_CircleCWRadius

Function **GMC_CircleCWRadius** executes a clockwise circular movement with a given centre.

Syntax:

```
ret_code := GMC_CircleCWRadius (ExecMode, ExecStatus.Interpld, EndSignal, Feed,  
                          Radius, AxisId, Position) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>Feed</i> (LREAL)	Speed in units/min
<i>Radius</i>	Radius with sign
<i>AxisId</i> (INT)	Axis identifier [1...64] for max 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters:

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (a TRUE)
-------------------------	--

Possible overload:

GMC_CircleCWRadius (INT,DWORD,INT,BOOL, FeedDescr_Struct,LREAL,
 INT,LREAL,INT,LREAL[,INT,LREAL]...)

Execution mode:

Wait / NoWait

Use:

This function executes a counter clockwise movement, which is either circular (if only two axes have been programmed) or helicoidal (more than 2 axes programmed), whose centre is located at a given point. The two axes that are programmed first are taken to be the abscissa and ordinate axes. At the end of the execution of the movement, signal “*EndSignal*” is set to true. Parameters “*AxisId*” and “*Position*” may be duplicated in order to request the synchronous movement of several axes, up to a maximum of 12 (2 as a minimum).

The movement is executed at the speed defined by the “*Feed*” parameter. This structure defines speed, acceleration, deceleration and jerk values which will be used during movement. If these fields are not set, the movement will be executed by default dynamics.

The following table describes the fields of the *Feed* structure.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Calculated speed only for linear axes

Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

Here follows the values that can be adopted by the *TypeFeed* variable.

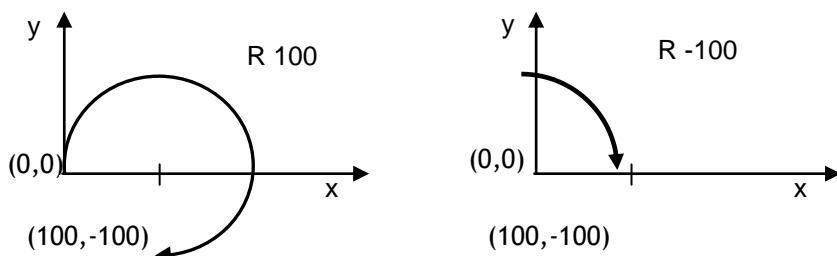
TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed rate and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Speed is to be conceived as vector speed, i.e., the rate of displacement of the set of axes moved, not that of the individual axes. Execution speed is limited based on the size of the radius to be executed, so as to limit the centrifugal acceleration that is generated by the circular motion. The limit is imposed by considering the acceleration of either the X or the Y axis, whichever is smaller.

$$\text{Feed} < \sqrt{A * \text{Radius}} \quad A = \min(\text{accX}, \text{accY})$$

The “Position” parameter may take on different meanings as a function of the axis programming context. In the case of absolute programming, this parameter refers to the point the axis will be at (end point) at end of movement, in the case of incremental programming it defines the distance (preceded by a sign) that the axis must cover starting from its current position. All end points are taken to refer to the origin activated for the programmed axis. If the operating limits are active, they are verified before executing the movement, and this check may generate an Over travel error.

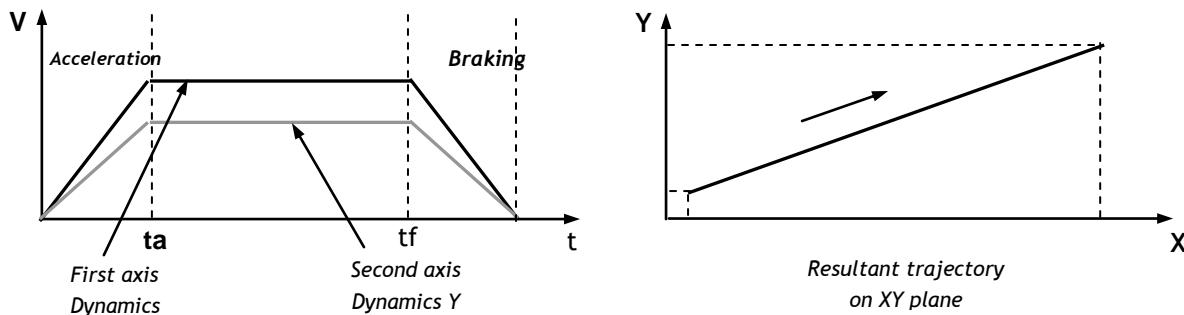
The “Radius” parameter has a sign which determines which of the two possible circles is going to be executed; if the sign is positive, the longer circle is executed, if it is negative, the shorter circle is selected.



This function may be programmed for form within the continuous stage

Coordinating the axes of a movement

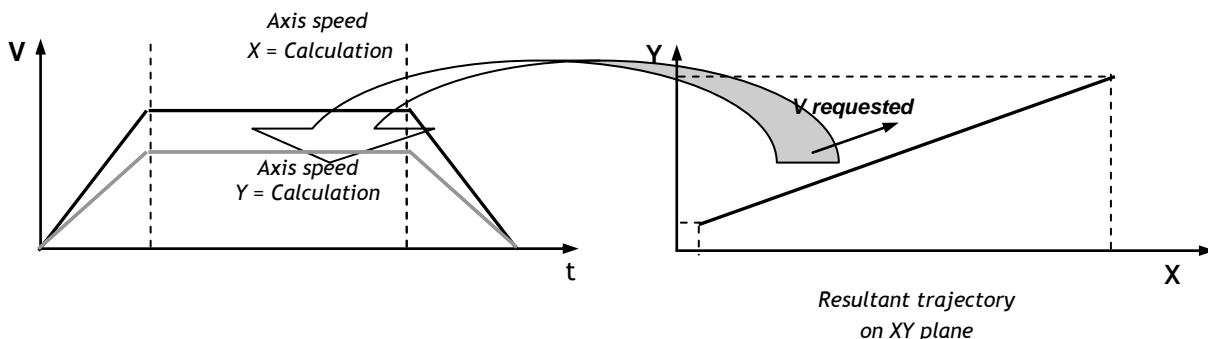
The axes defined in GMC_CircleCWRRadius start and end their movements simultaneously; hence they are mutually “interpolated” and therefore the required trajectory is maintained. From the dynamic standpoint each axis will adopt a speed curve of its own, independent of the other axes in terms of feed rates reached and accelerations and jerks adopted; since the motions of the various axes are mutually interpolated, the acceleration stage ends simultaneously for all the axes; similarly, the deceleration stage begins simultaneously.



In any event, checks are carried out to determine whether the speed curves requested for each axis exceed the (feed, acceleration and jerk) values configured for the axis in question. If the speed curves requested for an axis are too high, limits are set on the speed curves of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes with different clocks will have to be moved separately.

Programmed speed

This function works according to the principle of “speed on the profile”, i.e., the speed obtained for the movement as a whole; by this principle, the speed at which an axis moves is calculated based on speed on the profile and the motion component of the individual axis relative to the overall distance to be



covered "on the profile"

Circular motions

The programming of circular motions (also referred to as circumferences) is susceptible to programming inaccuracies (numerical representation) and hence the circles programmed are often rejected in as much as they do NOT describe a perfect circle. To remedy this problem, “tolerances” and “methodologies” are provided for use in defining the circles. In this connection, see function GMC_ToleranceParams.

Full circles

In programming a full circle, the end point must be made to coincide with the starting point. There is a system variable, **FullCirc (FCT)**, by which if the distance between the starting point and the end point is smaller than this constant, the circle executed will be a full circumference.

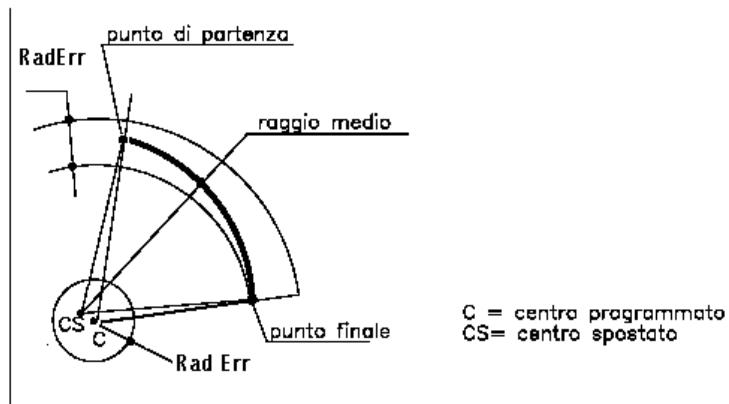
For example, assuming our circle has a starting point (100, 100) and an end point (100.1,100.1), it will be executed as an arc if FullCirc=0.1, and will be executed as a full circle (with end point 100,100) if FullCirc=0.15, the distance between the two points being 0.14142.

Circle execution modalities

Another problem to be addressed in connection with the programming of circular motions is the precision required in programming the centre in order to describe a perfect circumference. First of all, in programming a circumference, the radii subtended between the starting point and the end point are determined: if the differences between the radii is nil, the circle is executed as requested, if the difference exceeds a given threshold, as defined by system variable **RadErr (CET)**, an error is generated, if the difference is in the 0 - RadErr range, methodologies are provided for recalculating the circumference parameters so that the circumference may be defined in a congruent manner. Four different programming modes, defined by means of system variable **RadMode (ARM)**, are available, to:

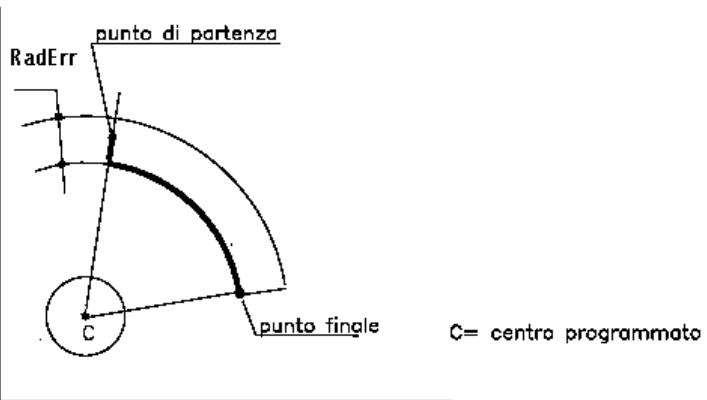
- ▷ Recalculate the centre of a circle within the tolerance defined by RadErr
- ▷ Move the starting point to within the tolerance defined by RadErr
- ▷ Recalculate the centre of a circle irrespective of RadErr (default mode)
- ▷ Maintain the centre where it is and move the starting and end points to within the tolerance defined by RadErr

RadMode =0 Recalculating the centre of a circle within a certain tolerance



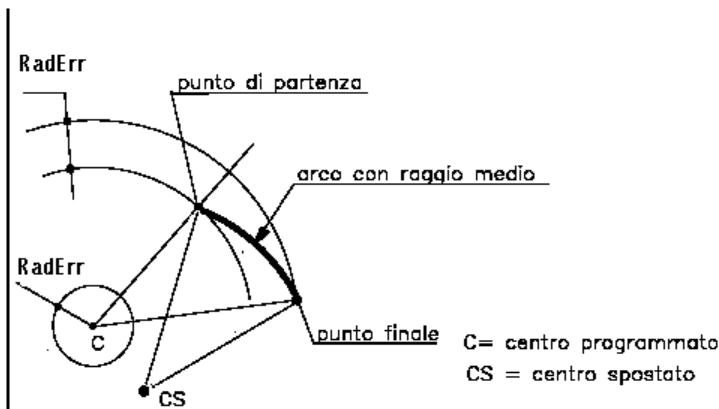
The circumference is recalculated so that it passes through the starting and end points and its centre is moved to within the tolerance range defined by parameter **RadErr**. In this case the circle is executed with a mean radius.

RadMode =1 Starting point moved to within a certain tolerance



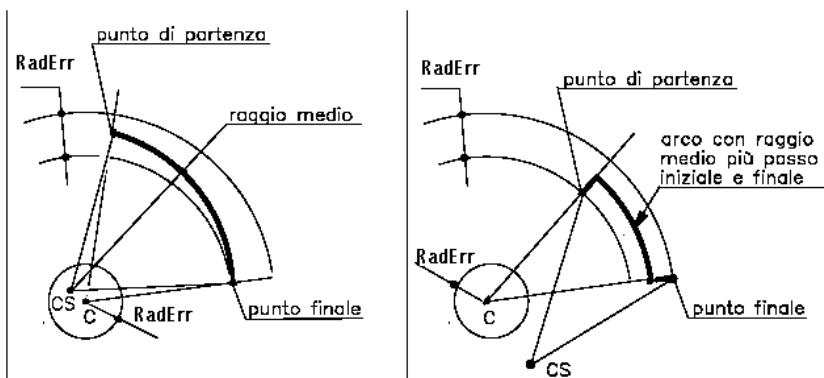
The circumference is recalculated so that it passes through a starting point moved to within the tolerance range defined by parameter **RadErr** and through the end point programmed. The centre is not moved.

RadMode =2 Recalculating the centre of a circle



RadMode =3 Keeping the centre where it is and moving the starting and end points

C = centro programmato
CS = centro spostato



If the movement of the centre of the circle takes place within the tolerance range defined by RadErr, then the circumference is recalculated so that it passes through the starting and end points and its centre is moved appropriately. (same as in RadMode=0). If the movement of the circle is not within the tolerance range specified by RadErr, then the centre is maintained where it is and the starting and end points are corrected by a value corresponding to RadErr/2

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160003	Too many activities in progress
0x00160014	Interpolator not present
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle

0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities have been used up
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(*Executes a clockwise circular movement at a feed rate of 10000
mm/min at point 100 for the first axis and point -100 for the second,
the centre is at -100.0, a 270° circle is executed.

At end of movement variable M0_0 is set to true*)
Feed.Value    := 10000.0 ;
Feed.TypeFeed := MODE_WORK;
Ret := GMC_CircleCWRRadius (MODE_WAIT, ULO, InterpId, M0_0, Feed,
                           -100.0, 1, 100.0, 2, -100.0) ;

```

See also:

[GMC_SetRamp](#), [GMC_SetJerk](#), [GMC_SetFeedOverride](#), [GMC_AbsoluteQuote](#), [GMC_IncrementalQuote](#), [GMC_ActivateOrigin](#), [GMC_ToleranceParms](#)

3.31 GMC_ContinuousAbort

Function GMC_ContinuousAbort determines the closure of a continuous stage and its cancellation.

Syntax:

```
ret_code := GMC_ContinuousAbort (ExecMode, ExecStatus, InterpId) ;
```

Input parameters

ExecMode (INT) Command execution mode [MODE_NOWAIT,
MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
ExecStatus (DWORD) Command execution status
InterpId (INT) Interpolator identifier PLC [1..40]

Execution mode:

Wait / NoWait

Use:

This function stops and cancels on a definitive basis the activities prepared during the continuous stage. Function GMC_ContinuousAbort puts an immediate end to and cancels all the movements saved.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not defined

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Closure and stop of continuous stage *)
Ret := GMC_ContinuousAbort (MODE_WAIT, ULO, InterpId) ;
```

See also:

[GMC_ContinuousStart](#), [GMC_ContinuousEnd](#), [GMC_ContinuousParam](#), [GMC_ContinuousParallel](#)

3.32 GMC_ContinuousEnd

Function GMC_ContinuousEnd determines the closure of a continuous stage.

Syntax:

```
ret_code := GMC_ContinuousEnd (ExecMode, ExecStatus, InterpId) ;
```

Input parameters:

ExecMode (INT) Command execution mode [MODE_NOWAIT,
MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
ExecStatus (DWORD) Command execution status
InterpId (INT) Interpolator identifier PLC [1..40]

Execution mode:

Wait / NoWait

Use:

This function closes the continuous movement saving stage. All the movements saved are started and at the end of their execution function GMC_ContinuousEnd ends.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not defined

Example:

```
Ret      : dword;
InterpId : INT;
...
(*Closure of continuous stage*)
Ret := GMC_ContinuousEnd (MODE_WAIT, UL0, InterpId) ;
```

See also:

[GMC_ContinuousStart](#), [GMC_ContinuousAbort](#), [GMC_ContinuousParam](#), [GMC_ContinuousParallel](#)

3.33 GMC_ContinuousParallel

Function GMC_ContinuousParallel activates continuous parallel motions.

Syntax:

```
ret_code := GMC_ContinuousParallel (ExecMode, ExecStatus, Interpld, MDA, StartCnd,  
                                  Count, AxisId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>MDA</i> (LREAL)	Angle to force speed to 0
<i>StartCnd</i> (INT)	Movement start mode
<i>Count</i> (INT)	Number of parallel motions
<i>AxisId</i> (INT)	Axis identifier [1..64]

Possible overload:

GMC_ContinuousParallel(INT,DWORD,INT,LREAL,INT,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This function starts the continuous parallel motion execution stage. It is necessary to specify all the axes (up to 12) involved in the subsequent continuous movements. In this manner, the axes will be immediately associated with the interpolator defined in the call and will not be available for use by any other interpolator until the continuous motion stage is over.

Unlike the GMC_StartContinuous function, this function involving parallel motions makes it possible to manage up to 12 parallel movements (continuous mode motion queues); the number of parallel movements is limited internally based on the number of axes acquired from the continuous mode.

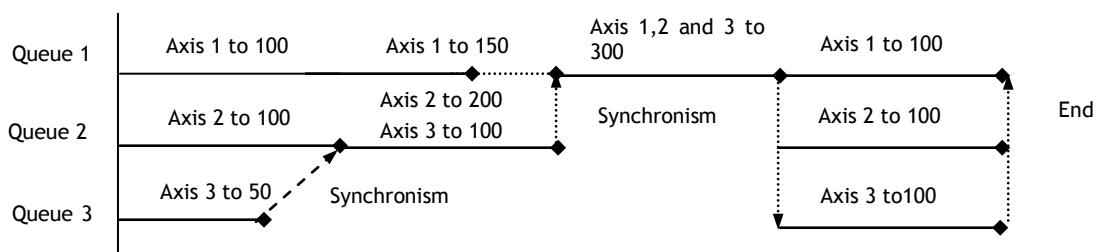
There is a policy governing the placement of the movements in one of the queues created for continuous parallel motions. This policy provides for the following rules:

- If there is a queue in which the last movement saved affects all the axes involved in the new movements, then the new movement is associated with the queue determined in this manner.
- If no axis is present in any queue, then the movement is associated either with a free queue or, if no queue is free, with the queue in which no movement has been placed over the longest period of time.
- If the axes involved in the movement are contained in different queues, the new movement is associated with the queue containing the first axis moved in the new movement, or the second (if the first axis is not moved in any queue) and so on.

Let's assume to activate a continuous movement with 3 parallel motions and recall movements on the axes with IDs 1, 2 and 3 according to the sequence defined below, and let us now analyse how these movements are distributed over the parallel motions:

Move	Description
ID 1 pos 100	Assigns movement to queue 1
ID 2 pos 100	Assigns movement to queue 2
ID 3 pos 50	Assigns movement to queue 3
ID 2 pos 200 ID 3 pos 100	Assigns movement to queue 2 and synchronises with queue 3
ID 1 pos 150	Assigns movement to queue 1
ID 1 pos 300 ID 2 pos 300 ID 3 pos 300	Assigns movement to queue 1 and synchronises with queue 2
ID 1 pos 100	Assigns movement to queue 1
ID 2 pos 100	Assigns movement to queue 2
ID 3 pos 100	Assigns movement to queue 3

If the movements in the example are distributed over a time diagram:



It is possible to decide whether the movement of the axes starts immediately or starts when the queue of the continuous movements has been filled up. This is done by means of the “*StartCnd*” parameter, which may be:

StartCnd =	0 CNT_NORM	Starts when queue has been filled or continuous stage ends
	1 CNT_START	Immediate starting

When an immediate start is requested, stops may occur between one movement and the next, since the system may be unable to calculate the subsequent movements soon enough.

In the continuous motion monitoring area, the progress of the continuous motion stage may be determined from the interpolator state that can be read through `GMC_InterpReadInfo`.

Description of continuous motions

Continuous motions are meant to eliminate as far as possible the stops between one movement and the next, by combining the different movements into a single one, as smooth as possible (from the dynamic standpoint). Slowdowns may be introduced between one movement and the next to enable the axes to go around possible sharp corners in the best way possible.

All the instructions recalled during the continuous stage are executed at once and return immediately to the caller. In particular, for movement functions, trajectory and dynamic calculations are performed, without starting the movement, however; the movements are saved in a queue on which the system will perform all the operations needed to optimise their execution (this optimisation stage is called **LookAhead**). Once enough motions have been grouped, the system will start the movement.

Axis movement begins upon the occurrence of one of the following events:

- › The LookAhead queue has been filled up. There is room for 64 elements in the queue, when the 64th element has been saved, the system starts the movement of the axes. As soon as one movement is completed and a queue element becomes available, a new movement instruction can be calculated and placed in the queue.
- › Lack of space to save new movements. The system admits a maximum number of movements to be distributed over the interpolators; assuming that different interpolators have activated movements, whether in continuous mode or not, all movements may turn out to have been used up. At this point the system will start the various movements in order to release the movements and get ready to acquire new ones.
- › Execution of a **GMC_MoveRoundOFF** instruction. The movement is activated immediately, regardless of the number of movements already saved. The **GMC_MoveRoundOFF** instruction will return to the caller as soon as the RoundOFF thresholds have been reached.
- › Execution of a **GMC_ContinuousEnd** instruction which determines the end of the movement saving stage.
- › Immediate start request, through the continuous stage activation or configuration functions.

LookAhead

The dynamic optimisation phase takes place in two steps:

- › dynamic optimisation of transition from one movement to the next
- › determination of stopping distance

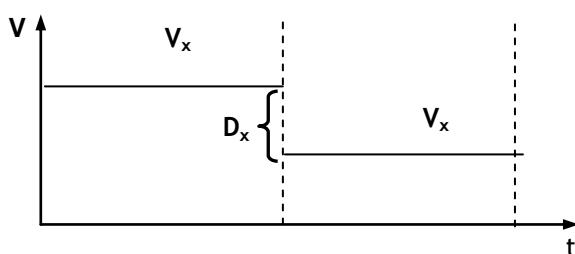
Dynamic optimisation is achieved by taking into account the motion components that a movement over an axis (speed) or several axes (direction and speed) may include. Let us assume there is an initial movement of length L_1 and a subsequent one of length L_2 , and both movements must be executed at speed V . In both movements, axis X is moved over a distance corresponding to X_1 during the first movement and X_2 during the second one. In both movements, the X axis will be moved at speeds determined according to the following formulas:

$$V_{x1} = V * X_1 / L_1 \quad \text{and} \quad V_{x2} = V * X_2 / L_2$$

Thus, in the transition from one movement to the next, the X axis will undergo an instantaneous speed variation of:

$$D_x = V_{x2} - V_{x1}$$

which might even be nil (if the L/X ratio remains constant).



This sudden change in speed might give rise to servo errors; accordingly, it is necessary to reduce this variation by reducing the speed at which the transition between one movement and the next is going to occur.

The slowdown applied can be controlled through an appropriate axis configuration parameter (**Maximum speed change**). In actual fact, the value of the speed change (Vc) will be calculated according to the following formula:

$$Vc = \text{Min} \left\{ \frac{\text{axis V change}}{|X1/L1 - X2/L2|}, V1, V2 \right\}$$

The speed at the transition point is affected by the MDA variable which defines (in degrees) the maximum angular deviation between one movement and the next. If the angle between two movements (angle in space for n axes) exceeds the value of the MDA variable, the transition speed between one entity and the next is set to 0.

Each time a movement is added to the continuous movement queue, the stopping distance calculation stage “recalculates” - for all the entities present in the continuous queue - the speeds at which the various movements may be executed so as to ensure a “controlled” stop, at least for the last movement contained in the queue. In this manner, we ensure that any event that might take place during the movement (Hold, emergency, reset) will be able to stop the axes. The greater is the number of entities contained in the queue, the higher is the speed that can be reached (albeit limited by the speed programmed for the movements) during axis movements.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160038	Continuous movement start condition wrong
0x00160014	Interpolator not defined
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities have been used up
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes of different clock

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Activates continuous stage for axes with ID 1 and ID 2 *)
Ret := GMC_ContinuousParallel (MODE_WAIT, ULO, InterpId, 90.0, 3,
                               1, 2, 3);
```

See also:

GMC_ContinuousEnd, GMC_ContinuousAbort, GMC_ContinuousParam, GMC_ContinuousStart

3.34 GMC_ContinuousParam

Function GMC_ContinuousParam changes continues mode movement parameters.

Syntax:

```
ret_code := GMC_ContinuousParam (ExecMode, ExecStatus, InterpId, Wait, Go, SplineIter) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>Wait</i> (INT)	Starting movement mode
<i>Go</i> (INT)	Starting movement mode
<i>SplineIter</i> (INT)	Reserved

Execution mode:

Wait / NoWait

Use:

This function makes it possible to change some continuous movement execution parameters. In particular, it is used to define or modify the following aspects:

Define whether or not to execute entry into tolerance at end of continuous motion:

Wait = 0 CNT_NOWAIT	Continuous stage without tolerance execution at closure
1 CNT_WAIT	Continuous stage without tolerance execution (habitual modality of continuous stage closure)

Change the immediate start modality of queued movements:

Go = 0 CNT_NORM	Starting at queue full or continuous stage closure
1 CNT_START	Immediate starting

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without error
0x00160038	Continuous motion start condition wrong
0x00160039	Tolerance management condition wrong
0x00160014	Interpolator not defined

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Starts movement in continuous mode *)
Ret := GMC_ContinuousParam (MODE_WAIT, ULO, InterpId, CNT_WAIT,
                           CNT_START, 0) ;
```

See also:

GMC_ContinuousEnd, GMC_ContinuousAbort, GMC_ContinuousStart, GMC_ContinuousParallel

3.35 GMC_ContinuousStart

Function GMC_ContinuousStart activates continuous motion.

Syntax:

```
ret_code := GMC_ContinuousStart (ExecMode, ExecStatus, InterpId, MDA, StartCnd, AxisId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>MDA</i> (LREAL)	Angle to force speed to 0
<i>StartCnd</i> (INT)	Movement start mode
<i>AxisId</i> (INT)	Axis identifier [1...64]

Possible overloads:

GMC_ContinuousStart(INT,DWORD,INT,LREAL,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This function starts the continuous motion execution stage. It is necessary to specify all the axes (up to 12) involved in the subsequent continuous movements. In this manner, the axes will be immediately associated with the interpolator defined in the call and will not be available for use by any other interpolator until the continuous motion stage is over.

It is possible to decide whether the movement of the axes starts immediately or starts when the queue of the continuous movements has been filled up. This is done by means of the “*StartCnd*” parameter, which may be:

StartCnd =	0 CNT_NORM	Starts when queue has been filled or continuous stage ends
	1 CNT_START	Immediate starting

When an immediate start is requested, stops may occur between one movement and the next, since the system may be unable to calculate the subsequent movements soon enough.

In the continuous motion monitoring area, from the interpolator state that can be read through GMC_InterpReadInfo, the progress of the continuous motion stage can be determined.

Description of continuous motions

Continuous motions are meant to eliminate to the extent feasible the stops between one movement and the next, by combining the different movements into a single one, as smooth as possible (from the dynamic standpoint). Slowdowns may be introduced between one movement and the next to enable the axes to go around possible sharp corners in the best way possible.

All the instructions recalled during the continuous stage are executed at once and return immediately to the caller. In particular, for movement functions, trajectory and dynamic calculations are performed, without starting the movement, however; the movements are saved in a queue on which the system will perform all the operations needed to optimise their execution (this optimisation stage is called **LookAhead**). Once enough motions have been grouped, the system will start the movement. Axis movement begins upon the occurrence of one of the following events:

- › The LookAhead queue has been filled up. There is room for 64 elements in the queue, when the 64th element has been saved, the system starts the movement of the axes. As soon as one movement is completed and a queue element becomes available, a new movement instruction can be calculated and placed in the queue.
- › Lack of space to save new movements. The system admits a maximum number of movements to be distributed over the interpolators; assuming that different interpolators have activated movements, whether in continuous mode or not, all movements may turn out to have been used up. At this point the system will start the various movements in order to release the movements and get ready to acquire new ones.
- › Execution of a **GMC_MoveRoundOFF** instruction. The movement is activated immediately, regardless of the number of movements already saved. The **GMC_MoveRoundOFF** instruction will return to the caller as soon as the RoundOFF thresholds have been reached.
- › Execution of a **GMC_ContinuousEnd** instruction which determines the end of the movement saving stage.
- › Immediate start request, through the continuous stage activation or configuration functions.

LookAhead

The dynamic optimisation phase takes place in two steps:

- › dynamic optimisation of transition from one movement to the next
- › determination of stopping distance

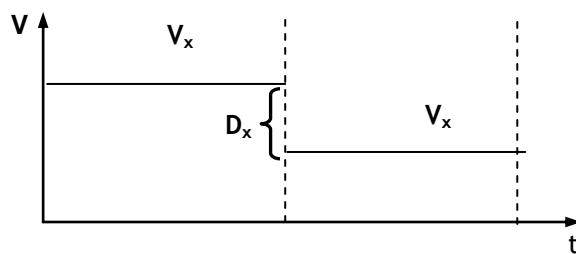
Dynamic optimisation is achieved by taking into account the motion components that a movement over an axis (speed) or several axes (direction and speed) may include. Let us assume there is an initial movement of length L_1 and a subsequent one of length L_2 , and both movements must be executed at speed V . In both movements, axis X is moved over a distance corresponding to X_1 during the first movement and X_2 during the second one. In both movements, the X axis will be moved at speeds determined according to the following formulas:

$$V_{x1} = V * X_1 / L_1 \quad \text{and} \quad V_{x2} = V * X_2 / L_2$$

Thus, in the transition from one movement to the next, the X axis will undergo an instantaneous speed variation of:

$$D_x = V_{x2} - V_{x1}$$

which might even be nil (if the L/X ratio remains constant).



This sudden change in speed might give rise to servo errors; accordingly, it is necessary to reduce this variation by reducing the speed at which the transition between one movement and the next is going to occur. The slowdown applied can be controlled through an appropriate axis configuration parameter (**Maximum speed change**). In actual fact, the value of the speed change (Vc) will be calculated according to the following formula:

$$Vc = \text{Min} \left\{ \frac{\text{axis V change}}{|X1/L1 - X2/L2|}, V1, V2 \right\}$$

The speed at the transition point is affected by the MDA variable which defines (in degrees) the maximum angular deviation between one movement and the next. If the angle between two movements (angle in space for n axes) exceeds the value of the MDA variable, the transition speed between one entity and the next is set to 0.

Each time a movement is added to the continuous movement queue, the stopping distance calculation stage “recalculates” - for all the entities present in the continuous queue - the speeds at which the various movements may be executed so as to ensure a “controlled” stop, at least for the last movement contained in the queue. In this manner, it is ensured that any event that might take place during the movement (Hold, emergency, reset) will be able to stop the axes. The greater is the number of blocks contained in the queue, the higher the speed that can be reached (albeit limited by the speed programmed for the movements) during axis movements.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160038	Continuous movement start condition wrong
0x00160014	Interpolator not defined
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities have been used up
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes of different clock

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Activates continuous stage for axes with ID 1 and ID 2 *)
Ret := GMC_ContinuousStart (MODE_WAIT, ULO, InterpId, 90.0, 1, 2) ;
```

See also:

GMC_ContinuousEnd, GMC_ContinuousAbort, GMC_ContinuousParam, GMC_ContinuousParallel

3.36 GMC_HwndExe

Function GMC_HwndExe activates the axes movement using an hand wheel.

Syntax:

```
ret_code := GMC_HwndExe (ExecMode, ExecStatus, Interpld, Tick, Steps, AxisId) ;
```

Input parameters:

ExecMode (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
ExecStatus (DWORD)	Command execution status
Interpld (INT)	Interpolator identifier PLC [1..40]
Tick (INT)	Cycle Time refresh hand wheel (in ms)
Steps(INT)	Hand wheel steps to run
Axis (INT)	Axes on which apply the hand wheel motion

Possible overload:

GMC_HwndExe (INT,DWORD,INT,INT,INT,[INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction enables the hand wheel motion and up to 12 axis identifiers can be associated with the function in order to manually move different axes at the same time. Motion is performed with the number of hand wheel *Steps* to make; this value is converted as a distance according to the parameter used when associating axis/hand wheel (*GMC_HwndParam*) that is:

$$\text{Space} = \text{Steps} * \text{Scale} / \text{Pitch}$$

The space value obtained will be added to the spaces previously signalled, in fact, PLC will cyclically notify the number of Steps to cover. *Tick* identifies the cycle time used by the PLC to refresh the steps. This parameter calculates the right axis speed rate limited by axis manual moves. A reverse motion stops the axis at null speed restarting its move in the opposite direction; the space not covered in the original direction is lost. If axis is homed, the software over travel limits will be activated during the hand wheel motion. The instruction can be sent at any time. To move the axis in hand wheel mode, “manual” move (see *GMC_MoveJog*) must be active. The *GMC_InterpReset* instruction stops the motion.

The correct instructions sequence to enable the move is:

<i>GMC_HwndParam</i>	Configuration of the hand wheel and its axis association
<i>GMC_HwndON</i>	Enabling hand wheel move
<i>GMC_MoveJog</i>	Move start
<i>GMC_HwndExe</i>	Move execution
<i>GMC_InterpReset</i>	Move end
<i>GMC_HwndOFF</i>	Disabling hand wheel move

Return values:

ret_code (HEX)	Description
0x00000000	Function execute without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160005	Too many axes in the process
0x0016002C	Axis ID not allowed
0x0016002B	Axis ID not found
0x0016000D	Interpolation activities exhausted
0x0016006C	Wrong Tick value
0x0016006B	Wrong steps value

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Performs hand wheel move of axes ID 1 and ID 2 refreshing every 50 ms, moves
16 steps *)
Ret := GMC_HwndExe (MODE_WAIT, ULO, InterpId, 50, 16, 1, 2) ;

```

See also:

[GMC_HwndON](#), [GMC_HwndParam](#), [GMC_HwndOFF](#), [GMC_MoveJog](#), [GMC_InterpReset](#)

3.37 GMC_HwndOFF

GMC_HwndOFF function disables the axes move by hand wheel.

Syntax:

```
ret_code := GMC_HwndOFF (ExecMode, ExecStatus, Interpld, AxisId) ;
```

Input parameters:

ExecMode (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
ExecStatus (DWORD)	Command execution status
Interpld (INT)	Interpolator identifier PLC [1..40]
Axis (INT)	Axis identifiers to disable hand wheel mode [1..64]

Possible overload:

GMC_HwndOFF (INT,DWORD,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction disables the hand wheel move for the axes specified in the function. It is possible to disable the move for up to 12 PLC axes. The instruction can be send at any time and if an hand wheel move is active, the axes are stopped at 0 speed and all the space left (represented by the function GMC_HwndExe) will be lost.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator missing
0x00160003	To many activities running
0x00160005	To many axes in the process
0x0016002C	Axis ID not allowed
0x0016002B	ID axis missing
0x0016000D	Interpolation activities finished

Example:

```
Result : dword;
InterpId : INT;
(* Hand wheel mode disabled for axes 1 and 2 *)
Result := GMC_HwndOFF (MODE_WAIT, UL2, InterpId, 1, 2 );
```

See also:

[GMC_HwndON](#), [GMC_HwndExe](#), [GMC_HwndParam](#)

3.38 GMC_HwndON

The GMC_HwndON function enables the hand wheel axes move.

Syntax:

```
ret_code := GMC_HwndON (ExecMode, ExecStatus, InterpId, AxisId) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterpId (INT)</i>	Interpolator identifier PLC [1..40]
<i>Axis (INT)</i>	Axis identifiers to disable hand wheel mode [1..64]

Possible overload:

GMC_HwndON (INT,DWORD,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This command disables the axes hand wheel mode. Up to 12 PLC axes can be moved. In order to make the function effective, axis/hand wheel configuration and matching have to be activated using the *GMC_HwndParam function*. The instruction can be sent at any time

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator missing
0x00160003	To many activities running
0x00160005	To many axes in the process
0x0016002C	Axis ID not allowed
0x0016002B	ID axis missing
0x0016000D	Interpolation activities finished

Example:

```
Result : dword;
InterpId : INT;

(* Axes 1 and 2 enabled for the hand wheel move*)
Result := GMC_HwndON (MODE_WAIT, UL2, InterpId, 1, 2 );
```

See also:

[GMC_HwndOFF](#), [GMC_HwndExe](#), [GMC_HwndParam](#)

3.39 GMC_HwndParam

Function GMC_HwndParam defines application parameters of the hand wheel on the axes.

Syntax:

```
ret_code := GMC_HwndExe (ExecMode, ExecStatus, InterpId, Scale, Pitch, AxisId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>Scale</i> (LREAL)	Distance covered in one hand wheel turn (mm/inches)
<i>Steps</i> (INT)	Steps number for one hand wheel revolution
<i>Axis</i> (INT)	Axes on which configure the hand wheel

Possible overload:

GMC_HwndParam (INT,WORD,INT,LREAL,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction associates the values to move the axis by hand wheel with the axes defined in the function. Up to 12 axes can be configured. The instruction can be sent at any time and the GMC_HwndON function enables the hand wheel axis move

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator missing
0x00160003	To many activities running
0x00160005	Too many axes in the process
0x0016002C	ID axis not admitted
0x0016002B	ID axis not found
0x0016000D	Interpolation activities finished
0x0016006B	Wrong Steps value
0x0016006A	Wrong Scale value

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Configures ID 1 and 2 axes for the hand wheel having
   5mm scale for 32 steps *)
Ret := GMC_HwndParam (MODE_WAIT, ULO, InterpId, 5.0, 32, 1, 2) ;
```

See also:

GMC_HwndON, GMC_HwndExe, GMC_HwndOFF

3.40 GMC_HwndParamExt

The function `GMC_HwndParamExt` defines the handwheel application parameters on the axes.

Syntax

```
ret_code := GMC_HwndExeExt (ExecMode, ExecStatus, InterpId, Scale, Pitch, Mode, Timeout,  
                  AxisId) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>Scale</i> (LREAL)	Distance to go for one handwheel round (mm/inches)
<i>Steps</i> (INT)	Steps number for a handwheel round
<i>Mode</i> (INT)	Handwheel management mode
<i>Timeout</i> (INT)	Timeout for immediate stop (ms)
<i>Axis</i> (INT)	Axes for which executing the handwheel configuration

Possible overload

GMC_HwndParamExt (INT,DWORD,INT,LREAL,INT,INT,INT,INT[,INT]...)

Execution mode

Wait / NoWait

Use

This command defines the values necessary to manage the defined axes Up to 12 axes can be configured. The command can be given at any time; after this command, in order to move the axis with handwheel, it is necessary to enable it via the GMC_HwndON function.

The parameter ***Mode***, enables special handwheel movement.

Mode = HWND_NORMAL (0) Uses all the pulses (*Steps*) entered from the handwheel for the movement.

Mode = HWND_TIMEOUT (1) With the parameter **Timeout** the timeout is managed on the pulses (*Steps*) sent for the movement. Once the timeout is enabled (in ms) there is an immediate request for axis stop.

Mode = HWND_OVERFEED (2) All the pulses (*Steps*) requiring an axis movement feed faster than the manual feed are ignored.

Return values

The following table lists all values *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160005	Too many axes involved
0x0016002C	Axis Id not accepted
0x0016002B	Axis Id not found
0x0016000D	Interpolation activities finished
0x0016006B	Steps value wrong
0x0016006A	Scale value wrong
0x00160085	Timeout value wrong
0x00160086	Mode value wrong

Example

```
Ret      : dword;
InterpId : INT;
...
(* Configures the axes with ID 1 and 2 for handwheel having 5mm scale for 32
steps *)
Ret := GMC_HwndParamExt (MODE_WAIT, UL0, InterpId, 5.0, 32, HWND_OVERFEED,
                           0.0, 1, 2) ;
```

See also

[GMC_HwndON](#), [GMC_HwndExe](#), [GMC_HwndOFF](#)

3.41 GMC_MSLdefine

Function GMC_MSLdefine executes the Master/Slave association.

Syntax:

```
ret_code := GMC_MSLdefine (ExecMode, ExecStatus, InterId, MasterId, SlaveId) ;
```

Input parameter:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	PLC interpolator identifier [1..40]
<i>MasterId (INT)</i>	Master axis identifier [1...64]
<i>SlaveId (INT)</i>	Slave axis identifier [1...64]

Possible overload:

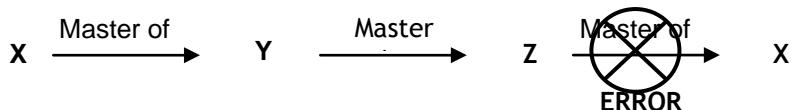
GMC_MSLdefine (INT,DWORD,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction associates (up to 12) slave axes with the master. This instruction **DOES NOT ENABLE** the **follower mode** and after this command the master does not move the slave. **This instruction allows the slave to be programmed as usual until the master follower mode is activated.** If a slave is already associated with a master, this function releases the previous association creates a new one and keeps the former following mode. If this operation is performed when the slave is already following the old master, the move is immediately extended to the new master A Master can be slave of another Master, defining a chain of Master/Slave axes. No error is accepted in the chain, as shown below:



A PLC interpolator has to be used to execute the Master/Slave management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator missing
0x00160003	Too many activities running
0x0016002C	Axis ID not accepted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Duplicated axis
0x001600C9	Master and slave cross-reference
0x0016002A	Axes with different clock

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Associates ID 1 Master with ID 2 and 3 slaves *)
Ret := GMC_MSLdefine (MODE_WAIT, UL0, InterpId, 1, 2, 3) ;

```

See also:

GMC_MSLexe, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLfollow,
 GMC_MSL_followONposition GMC_MSLfollowONsignal, GMC_MSLjog, GMC_MSLjogONposition,
 GMC_MSLjogONsignal, GMC_MSLparam. GMC_MSLreset, GMC_MSLresetONposition,
 GMC_MSLresetONsignal, GMC_MSLundef

3.42 GMC_MSLexe

Function GMC_MSLexe immediately activated the Master/Slave following.

Syntax:

```
ret_code := GMC_MSLexe (ExecMode, ExecStatus, InterId, Oper, SlaveId) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	PLC interpolator identifier [1..40]
<i>Oper (INT)</i>	Operation to execute
<i>SlaveId (INT)</i>	Slave axis identifier [1...64]

Possible overload:

GMC_MSLexe(INT,DWORD,INT,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction enables/disables the Slave following of the Master. If following is disabled, slave is kept associated with the master but stops to follow it. According to the *Oper* parameter, the following operations can be executed:

Oper =	0 MSL_GO	Enables following
	1 MSL_STOP	Disables following
	2 MSL_JOGING	Disables following maintaining its feed rate



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x0016002C	Axis ID not accepted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x001600CD	Wrong following mode
0x0016000D	Interpolation activities finished
0x00160028	Axis in use by another interpolator
0x00160029	Axis in use by another interpolator in Hold

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Enables slave ID 2 and 3 following *)
Ret := GMC_MSLexe (MODE_WAIT, UL0, InterpId, MSL_GO, 2, 3) ;
```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLfollow,
GMC_MSL_followONposition GMC_MSLfollowONsignal, GMC_MSLjog, GMC_MSLjogONposition,
GMC_MSLjogONsignal, GMC_MSLdefine. GMC_MSLreset, GMC_MSLresetONposition,
GMC_MSLresetONsignal, GMC_MSLundef

3.43 GMC_MSLexeONposition

Function GMC_MSLexeONposition activates the Master/Slave following on axis position.

Syntax:

```
ret_code := GMC_MSLexeONposition (ExecMode, ExecStatus, InterId, AxisId, Mode, Position,
                                  Oper, SlaveId) ;
```

Input parameters:

ExecMode (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
ExecStatus (DWORD)	Command execution status
InterId (INT)	PLC interpolator identifier [1..40]
AxisId (INT)	Axis ID [1...64] for starting synchronism
Mode (INT)	Moving start mode
Position (LREAL)	Axis position for starting synchronism
Oper (INT)	Operation to execute
SlaveId (INT)	Slave axis identifier[1...64]

Possible overload:

GMC_MSLexeONposition(INT,DWORD,INT,INT,INT,LREAL,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction enables/disables the Master/Slave following according to the axis position. If following mode is disabled, slave stops to follow the master but is still associated to it. *Oper* parameter executes the following actions:

Oper = 0 MSL_GO	Enables following
1 MSL_STOP	Disables following
2 MSL_JOGING	Disables following maintaining its feed rate

Instruction is executed if AxisID doesn't change *Mode* condition according to the *Position*, that is:

Mode = 10 CND_AXL	Starts when axis AxisId is at an interpolated position lower than Position. The position must take into account the offsets active on the AxisId axis.
11 CND_AXG	Starts when axis AxisId is at an interpolated position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis.
12 CND_AXLABS	Starts when axis AxisId is at an interpolated position lower than Position. This position must be construed as absolute.
13 CND_AXGABS	Starts when axis AxisId is at an interpolated position higher than or equal to Position. This position must be construed as absolute.
14 CND_AXLph	Starts when axis AxisId is at a real position lower than Position. The position must take into account the offsets active on the AxisId axis.

15	CND_AXGph	Starts when axis tAxisId is at a real position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis.
16	CND_AXLABSph	Starts when axis AxisId is at a real position lower than Position. This position must be construed as absolute.
17	CND_AXGABSpj	Starts when axis AxisId is at a real position higher than or equal to Position. This position must be construed as absolute.

The function terminates as soon as the instruction is executed.

Absolute/incremental programming mode is NOT applied on *Position* parameter, therefore the position is considered absolute only and the axis origins are taken into account. If *AxisId* is the master, the “special” origins related to Master/Slave are considered according to these rules:

- ▷ If all the slaves are associated to the *AxisId* master, they all have the same master origin value (GMC_MSLresetxxx function is applied to all slaves at the same time)
- ▷ If only some axes are associated to the *AxisId*, all slaves must have a master value origin of 0.0.



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not fund
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x001600CD	Wrong following mode
0x0016000D	Interpolation activities finished
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160036	Signal synchronism variable wrong
0x00160032	Wrong starting movement condition
0x00160005	Too many axes involved
0x0016005E	Wrong rollover axis position

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Enables slaves ID 2 and 3 following when ID 1 position is >= 300.0 *)
Ret := GMC_MSLexeOnPosition (MODE_WAIT, UL0, InterpId, 1, CND_AXG, 300.0,
                           MSL_GO, 2, 3) ;
```

See also:

[GMC_MSLparam](#), [GMC_MSLexeONsignal](#), [GMC_MSLexe](#), [GMC_MSLfollow](#), [GMC_MSL_followONposition](#)
[GMC_MSLfollowONsignal](#), [GMC_MSLjog](#), [GMC_MSLjogONposition](#), [GMC_MSLjogONsignal](#), [GMC_MSLdefine](#).
[GMC_MSLreset](#), [GMC_MSLresetONposition](#), [GMC_MSLresetONsignal](#), [GMC_MSLundef](#)

3.44 GMC_MSLexeONsignal

The **GMC_MSLexeONsignal** function activates Master/Slave following on signal.

Syntax:

ret_code := GMC_MSLexeONsignal (ExecMode, ExecStatus, InterId, Signal, Mode, Oper, SlaveId);

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	PLC interpolator identifier [1..40]
<i>Signal (BOOL)</i>	Boolean variable for starting synchronism
<i>Mode (INT)</i>	Moving start mode
<i>Oper (INT)</i>	Operation to execute
<i>SlaveId (INT)</i>	Slave axis identifier [1...64]

Possible overload:

GMC_MSLexeONsignal(INT,WORD,INT,BOOL,INT,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction enables/disables the Master following mode according to the signal status. If disabled, the slave stops to follow the master, but keeps on being associated to it. According to the *Oper* parameter the following operations can be performed:

<i>Oper =</i>	0 MSL_GO	Enables following
	1 MSL_STOP	Disables following
	2 MSL_JOGING	Disables following and proceeds in feed

Motion starts when “*Signal*” doesn’t change “*Mode*” condition:

<i>Mode =</i>	1 CND_RAISE	Start on rising edge of signal <i>Signal</i>
	2 CND_FAIL	Start on falling edge of signal <i>Signal</i>
	3 CND_ON	Start on “ON” level (true) of signal <i>Signal</i>
	4 CND_OFF	Start on “OFF” level (false) of signal <i>Signal</i>

The function finishes as soon as the instruction is executed.



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not fund
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x001600CD	Wrong following mode
0x0016000D	Interpolation activities finished
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160036	Signal synchronism variable wrong
0x00160032	Wrong starting movement condition
0x00160005	Too many axes involved

Example

```

Ret      : dword;
InterpId : INT;
...
(* Enables slave ID 2 and 3 following the rising edge of M0_0 *)
Ret := GMC_MSLexeOnSignal (MODE_WAIT, ULO, InterpId, M0_0, CND_RAISE,
                           MSL_GO, 2, 3) ;

```

See also:

[GMC_MSLparam](#), [GMC_MSLexeONposition](#), [GMC_MSLexe](#), [GMC_MSLfollow](#), [GMC_MSL_followONposition](#)
[GMC_MSLfollowONsignal](#), [GMC_MSLjog](#), [GMC_MSLjogONposition](#), [GMC_MSLjogONsignal](#), [GMC_MSLdefine](#).
[GMC_MSLreset](#), [GMC_MSLresetONposition](#), [GMC_MSLresetONsignal](#), [GMC_MSLundef](#)

3.45 GMC_MSLfollow

Function GMC_MSLfollow immediately modifies Master/Slave following ratio.

Syntax:

```
ret_code := GMC_MSLfollow (ExecMode, ExecStatus, InterpId, FollRate, SlaveId) ;
```

Input parameters:

<i>ecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterpId (INT)</i>	PLC interpolator identifier [1..40]
<i>FollRate (LREAL)</i>	Master following rate
<i>SlaveId (INT)</i>	Slave identifier [1...64]

Possible overload:

GMC_MSLfollow(INT,DWORD,INT,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction changes the Slave following ratio, restarting the synchronisation with the Master. **FollRate** value (slave following rate of the master) is a multiplier of the speed rate or of the space covered by the Master. If value is 1.0, slave will copy the Master motion, if value is lower or higher than 1.0. the slave increases/decreases its motion. This value can be signed.



A PLC interpolator has to be used to execute the Master/Slave management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not fund
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x0016000D	Wrong following mode

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Changes following for slave ID 2 and 3 *)
Ret := GMC_MSLfollow (MODE_WAIT, UL0, InterpId, 1.5, 2, 3) ;
```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONSignal, GMC_MSLexe,
GMC_MSL_followONposition GMC_MSLfollowONSignal, GMC_MSLjog, GMC_MSLjogONposition,
GMC_MSLjogONSignal, GMC_MSLdefine. GMC_MSLreset, GMC_MSLresetONposition,
GMC_MSLresetONSignal, GMC_MSLundef

3.46 GMC_MSLfollowONposition

Function GMC_MSLfollowONposition modifies Master/Slave following ratio on axis position.

Syntax:

```
ret_code := GMC_MSLfollowONposition (ExecMode, ExecStatus, Interpld, , AxisId, Mode, Position,
                                     FollRate, Slaveld) ;
```

Input parameters:

<i>ecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>Interpld (INT)</i>	PLC interpolator identifier [1..40]
<i>AxisId (INT)</i>	Axis ID [1...64] for starting synchronism
<i>Mode (INT)</i>	Moving start mode
<i>Position (LREAL)</i>	Axis position for starting synchronism
<i>FollRate (LREAL)</i>	Master following ratio
<i>Slaveld (INT)</i>	Slave identifier [1...64]

Possible overload:

GMC_MSLfollowONposition(INT,DWORD,INT,INT,INT,LREAL,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction changes the Slave following ratio of the Master according to an axis position. Synchronisation with the Master restarts. *FollRate* value (slave following ratio) is a multiplier of the speed rate or of the space covered by the Master. If value is 1.0, slave will copy the Master motion, if value is lower or higher than 1.0. the slave increases/decreases its motion. This value can be signed. The instruction is executed when *AxisId* doesn't change *Mode* condition according to the *Position*, that is:

Mode = 10 CND_AXL	Starts when axis AxisId is at an interpolated position lower than Position. The position must take into account the offsets active on the AxisId axis.
11 CND_AXG	Starts when axis AxisId is at an interpolated position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis.
12 CND_AXLABS	Starts when axis AxisId is at an interpolated position lower than Position. This position must be construed as absolute.
13 CND_AXGABS	Starts when axis AxisId is at an interpolated position higher than or equal to Position. This position must be construed as absolute.
14 CND_AXLph	Starts when axis AxisId is at a real position lower than Position. The position must take into account the offsets active on the AxisId axis.
15 CND_AXGph	Starts when axis tAxisId is at a real position higher than or

		equal to Position. The position must take into account the offsets active on the AxisId axis.
16	CND_AXLABSph	Starts when axis AxisId is at a real position lower than Position. This position must be construed as absolute.
17	CND_AXGABSpj	Starts when axis AxisId is at a real position higher than or equal to Position. This position must be construed as absolute.

The function finishes when the instruction is executed.

Absolute/incremental programming mode is NOT used for *Position* parameter therefore the position is considered absolute only. Anyway, axis origins are considered. If *AxisId* is the master of the instruction slaves, the “special” origins related to Master/Slave are considered according to the following rules:

- ▷ If all the slaves are associated to the master *AxisId*, they must have the same master origin value (i.e. GMC_MSLresetxxx function has been executed for all the axes at the same time).
- ▷ If only some axes are associated to the master *AxisId*, all the slaves need a master origin value of 0.0



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not fund
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x0016000D	Wrong following mode
0x00160032	Starting movement condition wrong
0x00160005	Too many axes involved
0x0016005E	Wrong rollover axis position

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Changes slaves ID 2 and 3 following when axis ID 1 is on 100.0 position *)
Ret := GMC_MSLfollowONposition (MODE_WAIT, UL0, InterpId, 1 CND_AXG,
                                100.0, 1.5, 2, 3) ;
```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLexe, GMC_MSL_follow
GMC_MSLfollowONsignal, GMC_MSLjog, GMC_MSLjogONposition, GMC_MSLjogONsignal, GMC_MSLdefine.
GMC_MSLreset, GMC_MSLresetONposition, GMC_MSLresetONsignal, GMC_MSLundef

3.47 GMC_MSLfollowONsignal

The GMC_MSLfollowONsignal function changes the Master/Slave following ratio on signal.

Syntax:

```
ret_code := GMC_MSLfollowONsignal (ExecMode, ExecStatus, InterId, , Signal, Mode,  
                  FollRate, Slaveld) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	PLC interpolator identifier [1..40]
<i>Signal</i> (BOOL)	Boolean variable for starting synchronism
<i>Mode</i> (INT)	Starting movement mode
<i>FollRate</i> (LREAL)	Master following rate
<i>Slaveld</i> (INT)	Slave identifier [1...64]

Possible overload:

GMC_MSLfollowONsignal(INT,DWORD,INT,BOOL,INT,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction changes the Slave following ratio of the Master according to an axis position. Synchronisation with the Master restarts. **FollRate** value (slave following ratio) is a multiplier of the speed rate or of the space covered by the Master. If value is 1.0, slave will copy the Master motion, if value is lower or higher than 1.0. the slave increases/decreases its motion. This value can be signed.

The instruction is executed when *Signal* is in *Mode* condition:

Mode =	1	CND_RAISE	Start on rising edge of signal <i>Signal</i>
	2	CND_FAIL	Start on falling edge of signal <i>Signal</i>
	3	CND_ON	Start on “ON” level (true) of signal <i>Signal</i>
	4	CND_OFF	Start on “OFF” level (false) of signal <i>Signal</i>

The function finishes when the instruction is executed.



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x0016000D	Interpolation activities finished
0x00160036	Signal synchronism variable wrong
0x00160005	Too many axis involved
0x00160032	Wrong starting movement condition

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Changes Slaves ID 2 and ID 3 following on signal M0_0 falling edge *)
Ret := GMC_MSLfollowONsignal (MODE_WAIT, ULO, InterpId, M0_0, CND_FAIL,
                               1.5, 2, 3) ;

```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLexe, GMC_MSL_follow
GMC_MSLfollowONposition, GMC_MSLjog, GMC_MSLjogONposition, GMC_MSLjogONsignal,
GMC_MSLdefine, GMC_MSLreset, GMC_MSLresetONposition, GMC_MSLresetONsignal, GMC_MSLundef

3.48 GMC_MSLjog

Function GMC_MSLjog immediately activates Jogging for a slave.

Syntax:

```
ret_code := GMC_MSLjog (ExecMode, ExecStatus, InterpId, JogVal, SlaveId) ;
```

Input parameters:

<i>ecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>JogVal</i> (LREAL)	Motion Feed (unit/min)
<i>SlaveId</i> (INT)	Slave identifier [1...64]

Possible overload:

GMC_MSLjog(INT,DWORD,INT,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction selects the speed rate of a slave not following the master yet (sign dictates the direction). The same command can change the speed rate or stop it (Speed=0). If the instruction is sent when slave is following the master, the following mode stops continuing at the speed required. Instruction is executed immediately.



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator missing
0x00160003	Too many activities running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Duplicated axis
0x0016000D	Interpolation activities finished

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Moves slave ID 2 and 3 with a 100.0 speed *)
Ret := GMC_MSLjog (MODE_WAIT, UL0, InterpId, 100.0, 2, 3) ;
```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONSignal, GMC_MSLexe,
GMC_MSL_followONposition, GMC_MSLfollowONSIGNAL, GMC_MSLfollow, GMC_MSLjogONposition,
GMC_MSLjogONSIGNAL, GMC_MSLdefine. GMC_MSLreset, GMC_MSLresetONposition,
GMC_MSLresetONSIGNAL, GMC_MSLundef

3.49 GMC_MSLjogONposition

Function GMC_MSLjogONposition immediately activates Jogging for a slave on axis position.

Syntax:

```
ret_code := GMC_MSLjogONposition (ExecMode, ExecStatus, InterId, AxisId, Mode, Position,
JogVal, SlaveId) ;
```

Input parameter:

ExecMode (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
ExecStatus (DWORD)	Command execution status
InterId (INT)	PLC interpolator identifier [1..40]
AxisId (INT)	Axis ID [1...64] for starting synchronism
Mode (INT)	Starting movement mode
Position (LREAL)	Axis position for starting synchronism
JogVal (LREAL)	Motion Feed (unit/min)
SlaveId (INT)	Slave identifier [1...64]

Possible overload:

GMC_MSLjogONposition(INT,DWORD,INT,INT,INT,LREAL,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction moves a slave axis not yet following the master (sign indicates the direction). The same command stops (Speed=0) or changes axis speed rate. If the instruction is sent while axis is following the master, the following ends switching to the required speed rate.

The instruction is executed when *AxisId* doesn't change the *Mode* condition according to the *Position* that is :

Mode = 10 CND_AXL	Starts when axis AxisId is at an interpolated position lower than Position. The position must take into account the offsets active on the AxisId axis.
11 CND_AXG	Starts when axis AxisId is at an interpolated position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis.
12 CND_AXLABS	Starts when axis AxisId is at an interpolated position lower than Position. This position must be construed as absolute.
13 CND_AXGABS	Starts when axis AxisId is at an interpolated position higher than or equal to Position. This position must be construed as absolute.
14 CND_AXLph	Starts when axis AxisId is at a real position lower than Position. The position must take into account the offsets active on the AxisId axis.
15 CND_AXGph	Starts when axis tAxisId is at a real position higher than or equal to Position. The position must take into account the offsets

		active on the AxisId axis.
16	CND_AXLABSph	Starts when axis AxisId is at a real position lower than Position. This position must be construed as absolute.
17	CND_AXGABSpj	Starts when axis AxisId is at a real position higher than or equal to Position. This position must be construed as absolute.

Function finishes as soon as the instruction is executed.

Absolute/incremental programming is NOT used for *Position* parameter, therefore the position is always considered as absolute even if the origins applied to the axis are taken into account. If *AxisId* is the master of the slaves within the instruction, also “special” origins related to Master/Slave are taken into account according to the following rules:

- ▷ If all slaves are associated to the *AxisId* master, they are required to have the same master origin value (i.e. function GMC_MSLresetxxx has been executed for all the slaves at the same time).
- ▷ If only some axes are associated to the master *AxisId*, all slaves are required to have a master origin value=0.0



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	To many activities running
0x0016002C	Unaccepted ID axis
0x0016002B	ID axis not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x0016000D	Interpolation activities finished
0x00160032	Starting movement condition wrong
0x00160005	Too many axes involved
0x0016005E	Wrong rollover axis position

Example:

```
Ret      : dword;
InterpId : INT;
...
(* When axis ID 1 exceeds 333.0, it moves the slave ID 3 with a 100.0 speed *)
Ret := GMC_MSLjogONposition (MODE_WAIT, UL0, InterpId, 1, CND_AXG, 333.0
                           100.0, 2) ;
```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLexe, GMC_MSL_follow
 GMC_MSLfollowONsignal, GMC_MSLfollowONposition, , GMC_MSLjog GMC_MSLjogONsignal,
 GMC_MSLdefine. GMC_MSLreset, GMC_MSLresetONposition, GMC_MSLresetONsignal, GMC_MSLundef

3.50 GMC_MSLjogONsignal

Function GMC_MSLjogONsignal enables Jogging for a slave on signal.

Syntax:

```
ret_code := GMC_MSLjogONsignal (ExecMode, ExecStatus, InterId, Signal, Mode,  
JogVal, SlaveId) ;
```

Input parameters:

ExecMode (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
ExecStatus (DWORD)	Command execution status
InterId (INT)	Identifier interpolator PLC [1..40]
Signal (BOOL)	Boolean variable for starting synchronism
Mode (INT)	Starting movement mode
JogVal (LREAL)	Motion Feed (unit/min)
SlaveId (INT)	Slave ID [1...64]

Possible overload:

GMC_MSLjogONsignal(INT,DWORD,INT,BOOL,INT,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction moves a slave axis not yet following the master (sign indicates the direction). The same command stops (Speed=0) or changes axis speed rate. If the instruction is sent while axis is following the master, the following mode ends switching to the required speed rate.

The instruction is executed if *Signal* doesn't change *Mode* condition:

Mode =	1	CND_RAISE	Start on rising edge of signal <i>Signal</i>
	2	CND_FAIL	Start on falling edge of signal <i>Signal</i>
	3	CND_ON	Start on "ON" level (true) of signal <i>Signal</i>
	4	CND_OFF	Start on "OFF" level (false) of signal <i>Signal</i>

Function finishes when instruction is executed.



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities running
0x0016002C	Unaccepted ID axis
0x0016002B	Id axis not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x0016000D	Interpolation activates finished
0x00160036	Signal synchronism variable wrong
0x00160005	Too many axes involved
0x00160032	Starting movement condition wrong

Example

```

Ret      : dword;
InterpId : INT;
...
(* Moves slave ID 2 at 100.0 speed on level ON of M0_0 *)
Ret := GMC_MSLjogONsignal (MODE_WAIT, UL0, InterpId, M0_0, CND_ON,
                           100.0, 2) ;

```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLexe, GMC_MSL_follow
GMC_MSLfollowONposition, GMC_MSLfollowONsignal, GMC_MSLjog, GMC_MSLjogONposition,
GMC_MSLdefine, GMC_MSLreset, GMC_MSLresetONposition, GMC_MSLresetONsignal, GMC_MSLundef

3.51 GMC_MSLparam

Function GMC_MSLparam defines Master/Slave axes following parameters.

Syntax:

```
ret_code := GMC_MSLparam (ExecMode, InterId, Mode, FollRate, SyncSpace, SlaveId) ;
```

Input parameters:

ExecMode (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
ExecStatus (DWORD)	Command execution status
InterId (INT)	Interpolator identifier PLC [1..40]
Mode (INT)	Following mode
FollRate (LREAL)	Master axis following rate
SyncSpace (LREAL)	Space required for the synchronization
SlaveId (INT)	Slave identifier [1...64]

Possible overload:

GMC_MSLparam(INT,DWORD,INT,INT,LREAL,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction defines the slave following mode of the master. The command has to be executed both before and during the following mode, in the latter case, values sent to the function will be activated during the next activation instruction **GMC_MSLe...** or will require a new master/slave synchronisation (**GMC_MSLfollow...**, **GMC_MSLdefine** where the master axis changes).

Mode parameter distinguishes the following modes:

Mode= 0	MSL_COPY	Follows the master copying its motions
1	MSL_VELO	Follows the master in velocity
2	MSL_SPC	Follows the master position
3	MSL_SPCVEL	Follows the master in position with approach space recovering
4	MSL_SPCVEL2	Follows the master in position with approach and release space recovering

SyncSpace value will be used only for MSL_VEL and MSL_POS following modes.

FollRate value is the slave following ratio and is a multiplier of the feed rate or of the space covered by the master. If value is 1.0, slave will copy the master motions, reducing or increasing it according to values lower or higher than 1.0. This value can be signed.



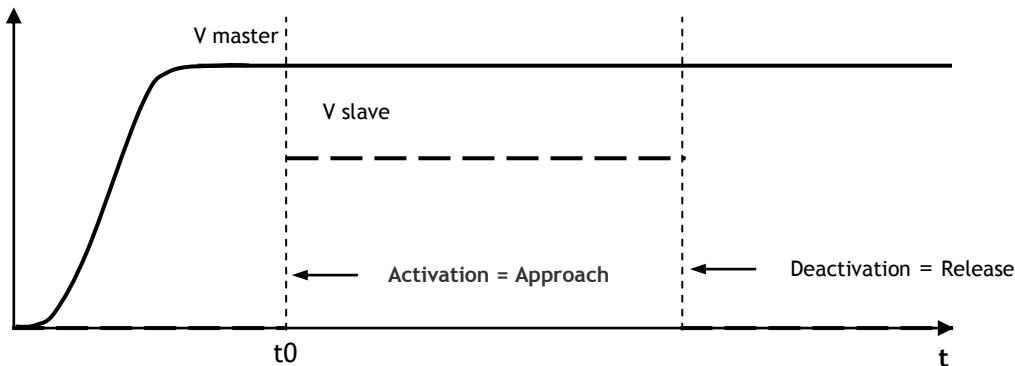
A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

MSL_COPY mode

In this mode the slave axis follow the master proportionately to the value of the ratio (if the ratio=1, the slave reproduces the movement of the master axis), synchronisation is instantaneous and the slave feed rate variation is “*in steps*”. Slave position and feed rate values are calculated, instant by instant, according to the following formulas:

$$V_{slave} = V_{master} * \text{FollowRate}$$

$$\text{PosSlave} = \text{PosSlave}_{t_0} + (\text{PosMaster} - \text{PosMaster}_{t_0}) * \text{FollowRate}$$

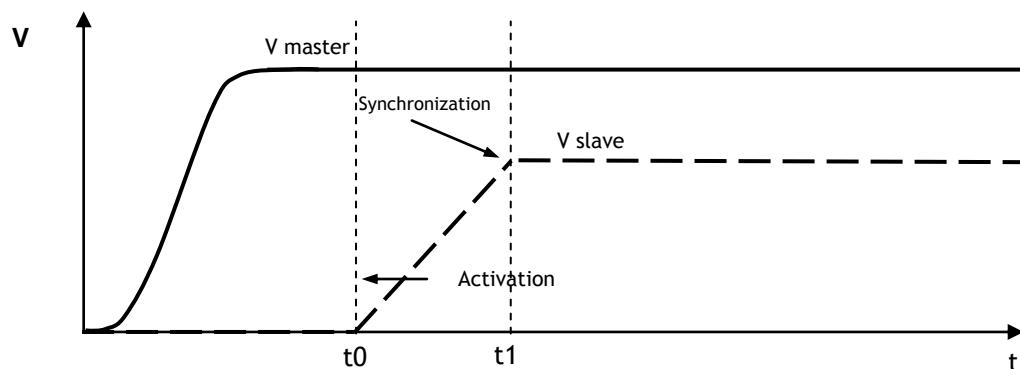


If the speed request for the slave axis exceeds its maximum admissible value, the system will reduce the feedrate requested accordingly giving out an emergency error (servo error) message, in that the slave is unable to follow the required position.

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero sped.

MSL_VELO mode

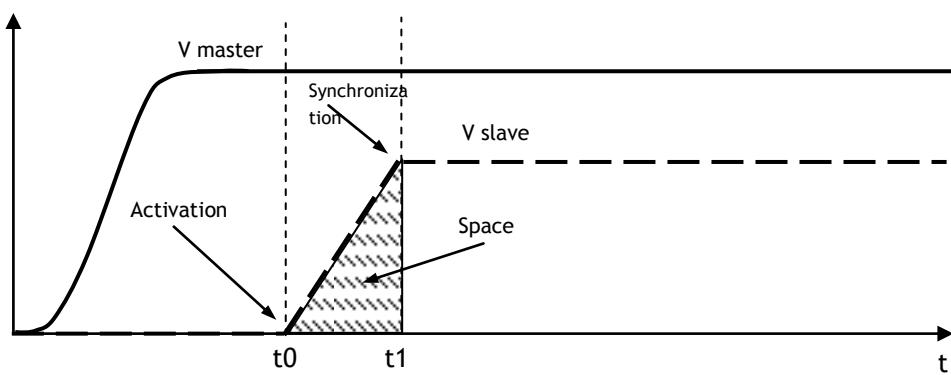
In this mode, the slave follows the feedrate of the master proportionately to the value of the ratio (if the ratio = 1, the slave copies the move of the master axis; the synchronisation depends on the dynamic characteristics of the slave axis and/or the Space parameter which defines the synchronisation distance. If distance = 0, the slave will synchronise with the master based on its maximum acceleration and using linear ramps only.



If values is not 0, the slave will synchronise with the master based on an acceleration calculated as a function of the synchronisation distance and using linear ramps only. The acceleration will be calculated again with each sampling process according to the formula below:

$$A_{slave} = ((V_{master} * \text{FollowRate})^2 + V_{slave}^2) / 2 * \text{Space}$$

where the value of the distance is gradually reduced based on the distance covered during the synchronisation stage. No check is made on the ensuing acceleration value, and therefore servo errors may occur if the acceleration exceeds the maximum value that can be withstood by the axis.



Once the synchronisation with the master has taken place, the slave will move according to this formula:

$$V_{slave} = V_{master} * FollowRate$$

The feed rate (V_{slave}) determined in this manner is “theoretical” since it is necessary to determine whether this request is compatible with the dynamic characteristics of the axis (maximum feed rate and maximum acceleration). The moment the master feed rate varies, the slave will follow this variation based on its own acceleration value. If the feed rate requested of the slave exceeds its maximum admissible feed rate, the system will reduce the feed rate requested accordingly. Hence, the feed rate and acceleration values with which the slave has to be moved, V_{slave_i} and A_{slave_i} , will be determined instant by instant. The position of the slave will therefore be calculated on the basis of these values:

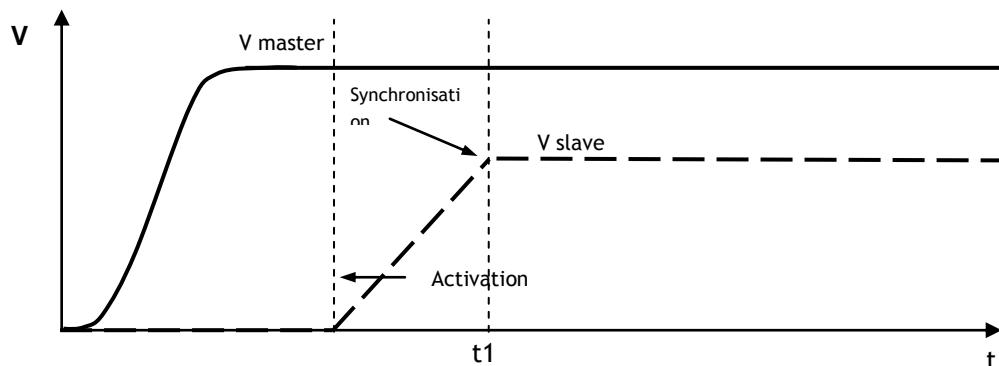
$$PosSlave_{tn+1} = PosSlave_{tn} + V_{slave_i} + A_{slave_i}$$

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed, using its deceleration ramp.

MSL_SPC mode

In this mode, the slave follows the position of the master proportionately to the value of the ratio (if the ratio = 1 the slave reproduces exactly the movement of the master); synchronisation depends on the dynamic characteristics of the slave axis and/or the Space parameter which defines the synchronisation distance.

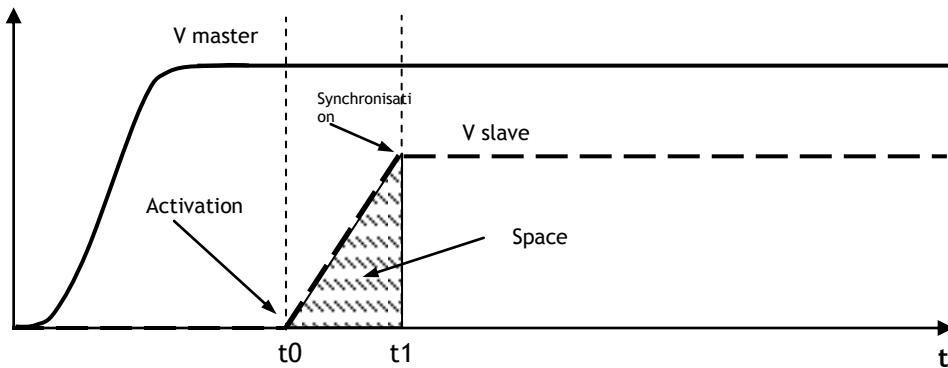
If distance is 0, the slave will synchronise with the master based on its maximum acceleration and using linear ramps **only**.



If the distance is not 0, the slave axis will synchronise with the master axis based on an acceleration calculated as a function of the synchronisation distance and using linear ramps **only**. The acceleration value will be calculated again with each sampling step according to this formula:

$$A_{slave} = ((V_{master} * FollowRate)^2 + V_{slave}^2) / 2 * Space$$

where the value of the distance is gradually reduced according to the distance covered during the synchronisation stage. No check is made on the ensuing acceleration value, and therefore servo errors may arise if the acceleration exceeds the maximum value that can be withstood by the axis.



Once the synchronisation with the master is made, the slave will move according to the following formulas:

$$\begin{aligned} PosSlave &= PosSlave_{t1} + (PosMaster - PosMaster_{t1}) * FollowRate \\ VSLAVE &= VMASTER * FOLLOWRATE \end{aligned}$$

The position, **PosSlave**, and the feed rate, **Vslave**, determined in this manner should be rated as "theoretical" values, since it is necessary to determine whether the values requested are compatible with the dynamic characteristics of the axis (Maximum feed rate and maximum acceleration). The moment the feed rate of the master varies, the slave will follow this variation according to its own acceleration value. If the feed rate requested for the slave exceeds the maximum value admissible for this axis, the system will reduce the feed rate accordingly. To this end, the two values with which to move the slave, **Vslave** and **Aslave**, will be calculated instant by instant. The actual position of the slave axis will therefore be calculated on the basis of these values:

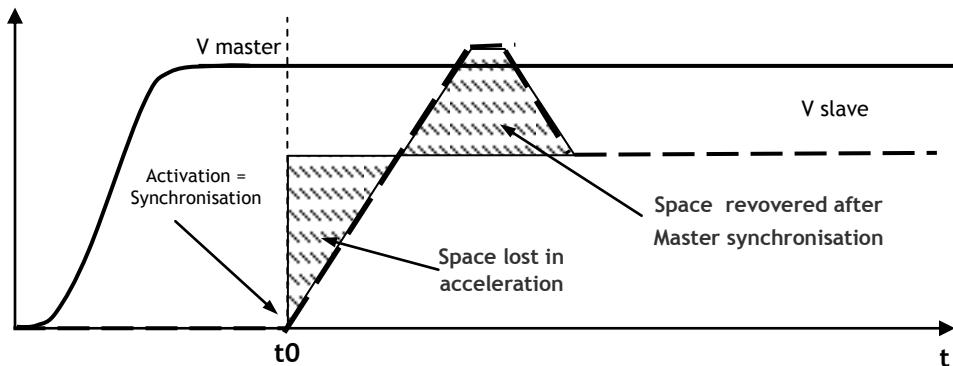
$$PosSlave_{tn+1} = PosSlave_{tn} + Vslave_i + Aslave_i$$

The difference between the actual and the theoretical position of the axis is taken up by the slave during its motion (even when the master has stopped moving) by moving, to the extent feasible, at a rate higher than the theoretical **Vslave**.

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed, using its deceleration ramp.

MSL_SPCVEL mode

In this mode, the slave follows the position and feedrate of the master proportionately to the value of the ratio (if ratio = 1, the slave reproduces the master move); synchronisation depends on the dynamic characteristics of the slave.



During the entire movement of the slave (i.e. both during and after the synchronisation stage), the axis moves according to the following formulas (always using linear ramps):

$$\text{PosSlave} = \text{PosSlave}_{t_0} + (\text{PosMaster} - \text{PosMaster}_{t_0}) * \text{FollowRate}$$

$$V_{SLAVE} = V_{MASTER} * FOLLOWRATE$$

The position (**PosSlave**) and the feed rate (**Vslave**) determined in this manner should be rated as "theoretical" values, in that it is necessary to determine whether these requests are compatible with the dynamic characteristics of the axis (max admissible feed rate and max admissible acceleration). The moment the feed rate of the master varies, the slave follows the variation according to its own acceleration value. If the feed rate requested of the slave is higher than its maximum admissible feed rate, the system reduces the feed rate requested accordingly. To this end, the two values with which the axis is to be moved (**Vslave_i** and **Aslave_i**) will be calculated instant by instant.

The actual position of the slave will therefore be calculated on the basis of these values:

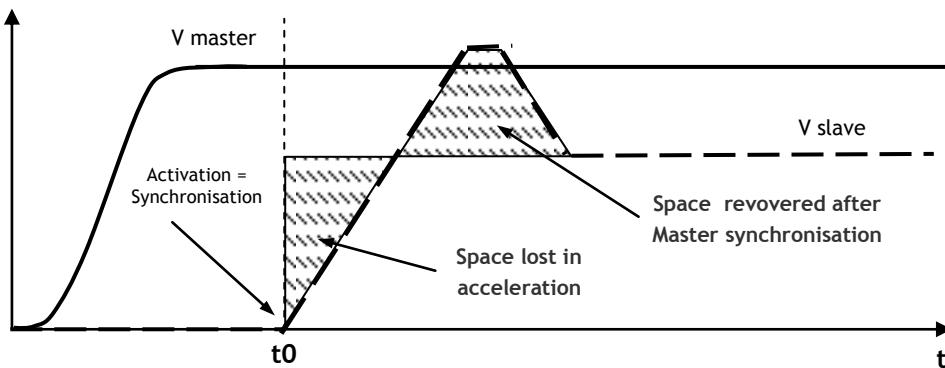
$$\text{PosSlave}_{t_{n+1}} = \text{PosSlave}_{t_n} + V_{slave\ i} + A_{slave\ i}$$

The difference between the actual and the theoretical position of the axis is taken up by the slave during its motion (even when the master has stopped moving) by moving, to the extent feasible, at a rate higher than the theoretical **Vslave**.

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed, using its deceleration ramp.

MSL_SPCVEL2 mode

In this mode, the slave follows the position and feed rate of the master proportionately to the value of the ratio (if the ratio = 1, the slave reproduces exactly the movement of the master); synchronisation depends on the dynamic characteristics of the slave.



During the entire movement of the slave (i.e. both during and after the synchronisation stage), the motion of the axis is according to the following formulas (always using linear ramps):

$$\text{PosSlave} = \text{PosSlave}_{t_0} + (\text{PosMaster} - \text{PosMaster}_{t_0}) * \text{FollowRate}$$

$$V_{SLAVE} = V_{MASTER} * FOLLOWRATE$$

The position (**PosSlave**) and the feed rate (**Vslave**) determined in this manner should be rated as “theoretical” values, in that it is necessary to determine whether these requests are compatible with the dynamic characteristics of the axis (max admissible feed rate and max admissible acceleration). The moment the feed rate of the master varies, the slave follows the variation according to its own acceleration value. If the feed rate requested of the slave is higher than its maximum admissible feed rate, the system reduces the feed rate requested accordingly. To this end, the two values with which the axis is to be moved (**Vslave_i** and **Aslave_i**) will be calculated instant by instant. The actual position of the slave will therefore be calculated on the basis of these values:

$$\text{PosSlave}_{t_{n+1}} = \text{PosSlave}_n + V_{slave\ i} + A_{slave\ i}$$

The difference between the actual and the theoretical position of the axis is taken up by the slave during its motion (even when the master has stopped moving) by moving, to the extent feasible, at a rate higher than the theoretical **Vslave**.

The moment the slave is released from the master, i.e., the moment the following mode is deactivated, the slave will slow down from its current speed to zero speed, using its deceleration ramp. Unlike the other modes, once it has reached zero speed, the master axis will reposition itself at the point where it was the moment the deactivation request was made.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activates running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not found
0x0016002D	ID axis related to a gantry
0x00160027	ID axis related to a spindle
0x0016002F	Duplicated mode
0x001600CD	Wrong following mode

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Defines slave ID 3 following of space+speed with 1.0 rate *)
Ret := GMC_MSLparam (MODE_WAIT, ULO, InterpId, MSL_SPCVEL, 1.0, 0.0, 3) ;

```

See also:

GMC_MSLexe, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLfollow,
 GMC_MSL_followONposition GMC_MSLfollowONsignal, GMC_MSLjog, GMC_MSLjogONposition,
 GMC_MSLjogONsignal, GMC_MSLdefine. GMC_MSLreset, GMC_MSLresetONposition,
 GMC_MSLresetONsignal, GMC_MSLundef

3.52 GMC_MSLreset

Function GMC_MSLreset immediately resets the value of the Master compared to the Slave.

Syntax:

```
ret_code := GMC_MSLreset (ExecMode, ExecStatus, Interpld, RollStep, Offset, Slaveld) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>Interpld (INT)</i>	Interpolator identifier PLC [1..40]
<i>RollStep (LREAL)</i>	Step to rollover the position
<i>Offset (LREAL)</i>	Offset applied to the position
<i>Slaveld (INT)</i>	Slave identifier [1...64]

Possible overload:

GMC_MSLreset(INT,DWORD,INT,LREAL,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction identifies an origin on the Master axis with reference to Slaves specified in the instruction. In addition to the managing of origins and distances resets, a special origin associated with all slaves is defined for the Masters. When the slave approaches its master, these origins will be taken into account. For example, if master is X and its slaves YZK, the function GMC_MSLreset (used to activate the origin) executed at different X positions can define a different origin for each slave.

Slave axis	X Position on MSLreset	Internal value. Origin on X associated to the Slave	X position seen from the slave when X is 1000
Y	100	-100	900
Z	340	-340	660
K	-100	100	1100

The origin value is reset when associating master and slave.

By means of the “Offset” parameter an offset may be associated with an axis during the reset operation. For example, if the “Offset” parameter is set to 50.0, when the position reset operation is activated, the master axis will move to position 50.0 instead of 0.0 (only for the specified slaves).

By means of the “RollStep” parameter it is possible to define the modulus to be applied during the axis position reset. For instance, if a rotary axis is at 3610 degrees and a function is executed with a “RollStep” of 360, the master axis will move to a position of 10 degrees instead of 0.0 (only for the specified slaves).



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities running
0x0016002C	Unaccepted ID axis
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis duplicated
0x0016000D	Interpolation activities finished

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Master homing for ID 2 and ID 3 slaves *)
Ret := GMC_MSLreset (MODE_WAIT, ULO, InterpId, 0.0, 0.0, 2, 3) ;
```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONSignal, GMC_MSLexe,
 GMC_MSL_followONposition, GMC_MSLfollowONSignal, GMC_MSLfollow, GMC_MSLjog,
 GMC_MSLjogONposition, GMC_MSLjogONSignal, GMC_MSLdefine, GMC_MSLresetONposition,
 GMC_MSLresetONSignal, GMC_MSLundef

3.53 GMC_MSLresetONposition

Function `GMC_MSLresetONposition` immediately resets the value of a Master compared to the Slave according to the axis position.

Syntax:

```
ret_code := GMC_MSLresetONposition (ExecMode, ExecStatus, Interpld, AxisId, Mode, Position
RollStep, Offset, Slaveld) ;
```

Input parameters:

<code>ExecMode (INT)</code>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<code>ExecStatus (DWORD)</code>	Command execution status
<code>Interpld (INT)</code>	Interpolator identifier PLC [1..40]
<code>AxisId (INT)</code>	Axis Id [1...64] for starting synchronism
<code>Mode (INT)</code>	Moving start mode
<code>Position (LREAL)</code>	Axis position for starting synchronism
<code>RollStep (LREAL)</code>	Step to rollover the position
<code>Offset (LREAL)</code>	Offset to apply on the position
<code>Slaveld (INT)</code>	Slave identifier [1...64]

Possible overload:

`GMC_MSLresetONposition(INT,DWORD,INT,INT,INT,LREAL,LREAL,LREAL,INT[,INT]...)`

Execution mode:

Wait / NoWait

Use:

This instruction identifies an origin on the Master axis with reference to Slaves specified in the instruction. In addition to the managing of origins and distances resets, a special origin associated with all slaves is defined for the Masters. When the slave approaches its master, these origins will be taken into account. For example, if master is X and its slaves YZK, the function `GMC_MSLreset` (used to activate the origin) executed at different X positions can define a different origin for each slave.

Slave axis	X position on MSLreset	Internal value, Origin on X associated to the Slave	X position seen from the slave when X is 1000
Y	100	-100	900
Z	340	-340	660
K	-100	100	1100

The origin value is reset when associating master and slave.

By means of the “*Offset*” parameter we may define an offset to be associated with an axis during the reset operation. For example, if the “*Offset*” parameter is set to 50.0, when the position reset operation is activated, the master axis will move to position 50.0 instead of 0.0 (only for the slaves specified).

By means of the “*RollStep*” parameter we may define the modulus to be applied during the axis position reset. For instance, if a rotary axis is at 3610 degrees and we execute a function with a “*RollStep*” of 360, upon the execution of the function, the master axis will move to a position of 10 degrees instead of 0.0 (only for the slaves specified).

The instruction is executed when *AxisId* doesn't change the *Mode* condition according to the *Position* that is:

Mode	Mnemonic	Description
10	CND_AXL	Sets Signal when axis AxisId is at an interpolated position lower than Position. The position must take into account the offsets active on the AxisId axis
11	CND_AXG	Sets Signal when axis AxisId is at an interpolated position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis
12	CND_AXLABS	Sets Signal when axis AxisId is at an interpolated position lower than Position. This position must be construed as absolute.
13	CND_AXGABS	Sets Signal when axis AxisId is at an interpolated position higher than or equal to Position. This position must be construed as absolute.
14	CND_AXLph	Sets Signal when axis AxisId is at a real position lower than Position. The position must take into account the offsets active on the AxisId axis.
15	CND_AXGph	Sets Signal when axis AxisId is at a real position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis.
16	CND_AXLABSpj	Sets Signal when axis AxisId is at a real position lower than Position. This position must be construed as absolute.
17	CND_AXGABSpj	Sets Signal when axis AxisId is at a real position higher than or equal to Position. This position must be construed as absolute.

Function finishes as soon as the instruction is executed.

Absolute/incremental programming is NOT used for *Position* parameter, therefore the position is always considered as absolute even if the origins applied to the axis are taken into account. If *AxisId* is the master of the slaves within the instruction, also “special” origins related to Master/Slave are taken into account according to the following rules:

- ▷ If all slaves are associated to the *AxisId* master, they are required to have the same master origin value (i.e. function GMC_MSLresetxxx has been executed for all the slaves at the same time).
- ▷ If only some axes are associated to the master *AxisId*, all slaves are required to have a master origin value=0.0



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	To many activities running
0x0016002C	Unaccepted ID axis
0x0016002B	ID axis not found
0x0016002D	Axis ID referred to a gantry
0x00160027	Axis ID referred to a spindle
0x0016002F	Axis duplicated
0x0016000D	Interpolation activities finished
0x00160032	Wrong starting move condition
0x00160005	Too many axes involved
0x0016005E	Wrong rollover axis position

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Resets Master position for slave with ID 2 on axis ID 1 position 333.0 *)
Ret := GMC_MSLresetONposition (MODE_WAIT, ULO, InterpId, 1, CND_AXG,
                               333.0, 0.0, 0.0, 2) ;

```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONSIGNAL, GMC_MSLexe, GMC_MSL_follow
 GMC_MSLfollowONSIGNAL, GMC_MSLfollowONposition, GMC_MSLjog, GMC_MSLjogONSIGNAL,
 GMC_MSLjogONposition, GMC_MSLdefine, GMC_MSLreset, GMC_MSLresetONSIGNAL, GMC_MSLundef .

3.54 GMC_MSLresetONsignal

Function GMC_MSLresetONsignal resets the Master value according to a Slave based on signal.

Syntax:

```
ret_code := GMC_MSLresetONsignal (ExecMode, ExecStatus, InterId, Signal, Mode,
                                  RollStep, Offset, SlaveId) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution state
<i>InterId (INT)</i>	PLC interpolator identifier [1..40]
<i>Signal (BOOL)</i>	Boolean variable for starting synchronism
<i>Mode (INT)</i>	Moving start mode
<i>RollStep (LREAL)</i>	Step to rollover the position
<i>Offset (LREAL)</i>	Offset to apply on the position
<i>SlaveId (INT)</i>	Slave identifier [1...64]

Possible overload:

GMC_MSLresetONsignal(INT,DWORD,INT,BOOL,INT,LREAL,LREAL,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction identifies an origin on the Master axis with reference to Slaves specified in the instruction. In addition to the managing of origins and distances resets, a special origin associated with all slaves is defined for the Masters. When the slave approaches its master, these origins will be taken into account. For example, if master is X and its slaves YZK, the function GMC_MSLreset (used to activate the origin) executed at different X positions can define a different origin for each slave.

Slave axis	X position on MSLreset	Internal value, Origin on X associated to the Slave	X position seen from the slave when X is 1000
Y	100	-100	900
Z	340	-340	660
K	-100	100	1100

The origin value is reset when associating master and slave.

By means of the “Offset” parameter we may define an offset to be associated with an axis during the reset operation. For example, if the “Offset” parameter is set to 50.0, when the position reset operation is activated, the master axis will move to position 50.0 instead of 0.0 (only for the slaves specified).

By means of the “RollStep” parameter we may define the modulus to be applied during the axis position reset. For instance, if a rotary axis is at 3610 degrees and we execute a function with a “RollStep” of 360, upon the execution of the function, the master axis will move to a position of 10 degrees instead of 0.0 (only for the slaves specified).

By means of the "Mode" and "Position" parameters we may define when the instruction is activated:

mode =	1 CND_RAISE	Reset on the rising edge of the signal Signal
	2 CND_FAIL	Reset on the falling edge of the signal Signal
	3 CND_ON	Reset on ON (true) level of the signal Signal
	4 CND_OFF	Reset on OFF (false) level of the signal Signal

Function finishes as soon as the instruction is executed.



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not fund
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Duplicated axis
0x0016000D	Interpolation activities finished
0x00160036	Signal synchronism variable wrong
0x00160005	Too many axes involved
0x00160032	Wrong starting movement condition

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Resets Master position for slaves ID 2 on M0_0 level ON *)
Ret := GMC_MSLresetONsignal (MODE_WAIT, ULO, InterpId, M0_0, CND_ON,
                           0.0, 0.0, 2) ;
```

See also:

GMC_MSLparam, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLexe, GMC_MSL_follow
GMC_MSLfollowONposition, GMC_MSLfollowONsignal, GMC_MSLjog, GMC_MSLjogONsignal,
GMC_MSLjogONposition, GMC_MSLdefine, GMC_MSLreset, GMC_MSLresetONposition, GMC_MSLundef

3.55 GMC_MSLundef

Function GMC_MSLundef executes the dissociation between Master/Slave axes.

Syntax:

```
ret_code := GMC_MSLundef (ExecMode, ExecStatus, InterpId, SlaveId) ;
```

Input parameters:

<i>ecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterpId (INT)</i>	Interpolator identifier PLC [1..40]
<i>SlaveId (INT)</i>	Slave identifier [1...64]

Possible overload:

GMC_MSLundef (INT,DWORD,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

This instruction releases the association Master/Slaves (up to 12): from now on slaves can be moved independently. If Slave is following its master or is in jogging mode, error message comes out: slave can't move.



A PLC interpolator has to be used to execute the Master/Slave following management. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities running
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not fund
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Duplicated axis
0x001600CA	Not slave axis
0x001600CB	Axis still following the Master
0x001600CD	Axis with commands still active

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Releases slaves ID 2 and 3 from the master *)
Ret := GMC_MSLundef (MODE_WAIT, UL0, InterpId, 2, 3) ;
```

See also:

GMC_MSLexe, GMC_MSLexeONposition, GMC_MSLexeONsignal, GMC_MSLfollow,
GMC_MSL_followONposition GMC_MSLfollowONsignal, GMC_MSLjog, GMC_MSLjogONposition,
GMC_MSLjogONsignal, GMC_MSLparam. GMC_MSLreset, GMC_MSLresetONposition,
GMC_MSLresetONsignal, GMC_MSLdefine

3.56 GMC_MSLrealign

The function GMC_MSLrealign immediately realigns the position of a Slave with reference to the Master.

Syntax

```
ret_code := GMC_MSLrealign (ExecMode, ExecStatus, InterId, SlaveId) ;
```

Parametri di ingresso

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	PLC interpolator identifier [1..40]
<i>SlaveId</i> (INT)	Slave axis identifier [1...64]

Possible overloads

GMC_MSLrealign (INT,DWORD,INT,INT[,INT]...)

Execution mode

Wait / NoWait

Use

This function must be used in case the Slave axis position changes compared to its Master. The position change can be done directly from the PLC (for example ZeroShift movement) or indirectly via disabling and then axes enabling.

Return values

The following table lists all the values the *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x0016002C	Unaccepted axis Id
0x0016002B	Axis Id not found
0x0016002D	Axis Id referred to a gantry
0x00160027	Axis Id referred to a spindle
0x0016002F	Axis duplicated
0x0016000D	Interpolation activities finished

Example

```
Ret      : dword;
InterpId : INT;
...
(* Realigns the Slave position of ID 2 and 3 *)
Ret := GMC_MSLrealign (MODE_WAIT, UL0, InterpId, 2, 3);
```

See also

GMC_MSLexeONposition, GMC_MSLexeONSignal, GMC_MSLexeGMC_Probe

3.57 GMC_Probe

Function GMC_Probe executes probing cycle.

Syntax:

```
ret_code := GMC_Probe      (ExecMode, ExecStatus, InterId, EndSignal, Approach,
                           SafeDist, ProbDist, ProbRad, ProbFeed, Mode, Feed, AxisId, Position) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	Interpolator identifier PLC [1..40]
<i>SafeDist</i> (LREAL)	Safe distance
<i>ProbDist</i> (LREAL)	Probing distance
<i>ProbRad</i> (LREAL)	Probing radius
<i>ProbFeed</i> (LREAL)	Probing feed rate
<i>Mode</i> (INT)	Probing mode
<i>AxisId</i> (INT)	Axis Id [1...64]
<i>Position</i> (LREAL)	Axis probing position
<i>Feed</i> (FeedDescr_Struct)	Feed rate

Output parameters:

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (TRUE)
<i>Approach</i> (BOOL)	Approaching execution signal

Possible overload:

GMC_Probe	(INT, DWORD, INT, BOOL, BOOL, LREAL, LREAL, LREAL, LREAL, INT, FeedDescr_Struct, INT, LREAL[, INT, LREAL]....)
-----------	---

Execution mode:

Wait / NoWait

Use:

The “Position” parameter may have different meanings depending on the axis programming context. In the case of absolute programming, it specifies the final point where the axis will be at end of movement; in the case of incremental programming, it defines the distance (with sign) that the axis must cover starting from its current position. At any rate, all points have to be construed as relative to the origin activated on the axis programmed. Before a movement is executed, the relative operating limits, if active, are verified and may generate an Over travel error.



Function cannot be programmed within a continuous stage.

Probing cycle execution

Cycle execution (accelerations/decelerations, jerks and minimum ramp times) depends forform the structure defined by input Feed. This structure defines feed rate, acceleration, deceleration and jerk used in the motion. If these fields are left blank, motion will be executed by default dynamic.

Feed structure fields as follow:

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Speed calculated for linear axes only

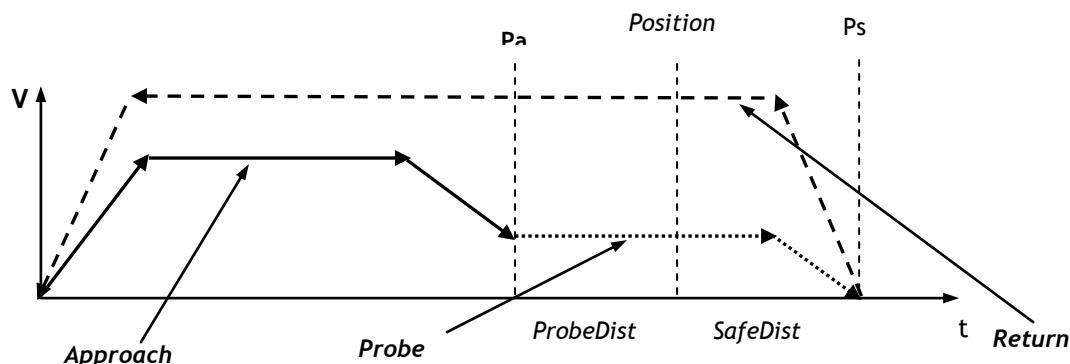
Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

Below all the possible values for the *TypeFeed* variable:

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Probing cycle stages can be described as follow:



- ▶ **Approach:** movement from current to approach point **Pa**; this point is the result of the difference between theoretical probing Position and the probe distance *ProbeDist* supplied as function parameter. Unless differently specified in the Feed programming structure, the motion is at rapid feed using working dynamic parameters. Speed rate is considered as vector, which is the feed rate of all moved axes and not of the single ones.
- ▶ **Probe:** motion is executed at probe feed rate (*ProbFeed*) up to the theoretical probe point (*Position*). Motion stops when all cycle axes probed, but the values memorized refers to the point each axis probed. If probing is not performed, axes move to safe point **Ps** and stop; this point results summing the theoretical probing point (*Position*) and the safe distance *SafeDist* (considered as function parameter). All dynamic parameters (accelerations/decelerations, Jerk and minimum ramp times) are involved in the motion. Speed is considered as vector, which is the speed rate of all axes moved and not of each single axis. If probe is not performed, axes will stop at the safe point **Ps**; this point is the result of the sum between *Position* (probing theoretical point) and *SafeDist* (the safe distance considered as function parameter). The motion is executed with all the dynamic parameters referring to accelerations/decelerations, jerks and minimum working motions. Velocity is considered as vector velocity that is the feed rate of all the axes moved and not of the single axes
- ▶ **Return:** is the optional stage depending on the *Mode* parameter. Unless Feed is programmed differently, if return is active, axes go back to the start point of the probing cycle with rapid feed and working dynamic parameters.

Mode	Mnemonic	Description
0	TCH_NORMAL	Executes the return stage
1	TCH_STOP	Does not execute the return stage

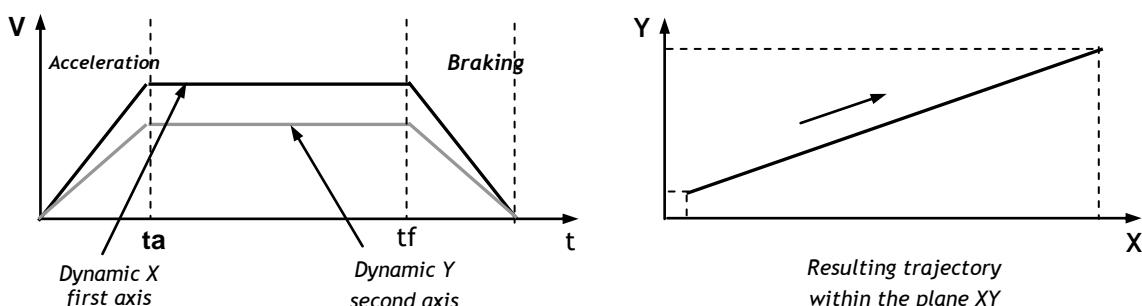
The result and the probe stage can be checked axis by axis within the axis status word (see `GMC_ReadAxisInfo`) or it can be read by `AX_ReadProbeStatus` function. The probed value has to be read by function `AX_ReadProbePosition`.



Probed value DOESN'T consider the probing radius (*ProbRad* parameter), but refers to the probing centre. *ProbRad* is used only to calculate probing positions.

Coordinating the axes of a movement

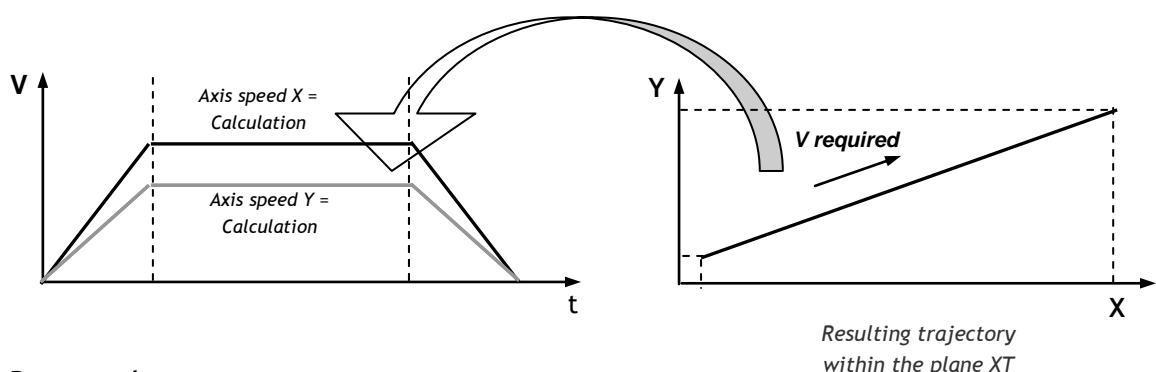
The axes defined in `GMC_Move` start and end their movements simultaneously; this means that they are “interpolated” with one another and hence the trajectory specified for the movement will be maintained. From the dynamic standpoint, each axis adopts a speed curve of its own, regardless of the other axes in terms of feed rate reached and acceleration and jerk values; since the motions of the various axes are mutually interpolated, the acceleration stage ends at the same time for all the axes: similarly, the deceleration stage starts simultaneously for all axes.



The system makes sure that the dynamic parameters requested for each axis do not exceed the (feed, acceleration and jerk) values programmed for it. If the dynamic values requested for an axis are exceedingly high, the system will limit the dynamics of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes having different clocks must be moved separately.

Programmed speed

This function works using the “speed on the profile”, i.e., resultant speed on the global movement; in this case the speed at which an axis moves will be determined starting from the speed on the profile and the component of the movement of the individual axis relative to the overall distance to be covered “on the profile”. As a rule, movements are performed at the highest possible feed rate, which is determined as a function of the maximum rate that each individual axis can reach. The speed obtained over a trajectory therefore depends both on the maximum feed that an axis can reach and on the extent to which an axis moves relative to global motion (ratio between the distance covered by the axis and the distance covered by the global movement).



Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	To many activities running
0x00160004	Move elements finished
0x00160005	Too many axes involved
0x0016002C	Id axis not accepted
0x0016002B	Id axis not found
0x0016002D	Id axis referred to a gantry
0x00160027	Id axis referred to a spindle
0x0016002F	Axis Id duplicated
0x0016000D	Interpolation activities finished
0x00160028	Axis already in use form another interpolator
0x00160029	Axis already in use form another interpolator in Hold
0x00160030	Axis already in use as slave form another interpolator
0x0016002A	Axes with different clocks
0x00160040	Axis on positive over travel
0x00160041	Axis on negative over travel
0x0016003C	Axis exceeds positive Software limit
0x0016003D	Axis exceeds negative Software limit
0x00160046	Null speed

ret_code (HEX)	Description
0x00160036	Wrong EndSignal or Approach synchronism variable errata
0x00160052	Wrong safe distance, value must be > 0
0x00160053	Wrong approaching distance, value must be >= 0
0x00160054	Wrong probing feedrate, values must be > 0
0x00160059	Wrong probing radius, value must be >= 0
0x0032005B	Option A48 not enabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(* Executes probing of the ID 1 axis at 100.0 position with V 50.0 *)
Ret := GMC_Probe (MODE_WAIT, UL0, InterpId, M0_0, M0_1, 5.0, 3.0, 0.0,
                  50.0, TCH_NORMAL, 1, 100.0, Feed) ;

```

See also:

[GMC_ReadAxisInfo](#), [AX_ReadProbeStatus](#), [AX_ReadProbePosition](#).

3.58 GMC_ProtectAreaDefine

GMC_ProtectAreaDefine function defines a protected area.

Syntax

```
ret_code:= GMC_ProtectAreaDefine (ExecMode, ExecStatus, Interpld, IdArea, AxisId, Low, Up);
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	PLC interpolator identifier [1..40]
<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Low</i> (LREAL)	Lower limit position
<i>Up</i> (LREAL)	Upper limit position

Output parameters

<i>IdArea</i> (INT)	Protected area id
---------------------	-------------------

Possible overloads

GMC_ProtectAreaDefine (INT,DWORD,INT,INT,INT,LREAL,LREAL[,INT,LREAL,LREAL]...)

Execution mode

Wait / NoWait

Use

This function defines a protected area. The area is made by axes having lower (*Low*) and upper (*Up*) activation limits. When 2 axes are defined, the protected area will trace a rectangle, if 3 axes are defined, the area will trace a parallelepiped and so on.

The function will return in the *IdArea* a number to be used in following area management calls(see GMC_ProtectAreaCmd) to enable, disable or delete the area. The area defined by this call is **inactive**.

Return values

The following table resumes all the *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160005	Too many axes involved
0x0016002C	Invalid axis id
0x0016002B	Axis id not found
0x0016002D	Axis id referred to a gantry
0x00160027	Axis id referred to a spindle
0x0016002F	Axis id duplicated
0x0016007F	Too many protected area defined

Example

```
Ret      : dword;
InterpId : INT;
AreaId   : INT;
...
(* Defines a protected parallelepiped in X(id 1)Y(id 2)Z(id 3) *)
Ret := GMC_ProtectAreaDefine (MODE_WAIT, UL0, InterpId, AreaId,
                           1,100.0,200.0, 2,20.0,50.0, 3,-100.0,0.0) ;
```

See also

[GMC_ProtectAreaCmd](#)

3.59 GMC_ReadAxisInfo

Function GMC_ReadAxisInfo reads the axis status.

Syntax:

```
ret_code := GMC_ReadAxisInfo (AxisId, Info) ;
```

Input parameters:

AxisId (INT) Axis identifier [1...64]

Output parameters:

Info (AxisInfo_Struct) Axis information structure

Possible overload:

GMC_ReadAxisInfo (INT,AxisInfo_Struct [,INT, AxisInfo_Struct]...)

Execution mode:

Immediate

Use:

“*AxisId*” and “*Info*” can be duplicated in order to ask synchronous axis status reading for up to 12 axes. The data associated with an axis are loaded into a type AxisInfo_Struct structure. If the axis requested is not present, the structure is loaded with data set to 0; if the ID field within the structure is set to 0, the axis requested is non-existent. The data contained in this structure are:

Field	Type	Description
AxisSwSts	DWORD	Axis software status (Motion)
AxisHwSts	DWORD	Axis hardware status
RealtSts	DWORD	Axis real time status
SharingProc	DWORD	Processes sharing the axis
MslStatus	WORD	MasterSlave status
PrbStatus	WORD	Probing status
SpnStatus	WORD	Spindle activity status
VirtStatus	WORD	Virtualizations status
AxisType	WORD	Axis type
Clock	WORD	Axis clock in 0,1 ms
Name	BYTE	Axis name
Id	BYTE	Axis ID
ActGear	BYTE	Active Gear
MasterId	BYTE	Master ID if axis is slave
RollPitch	LREAL	Rollover pitch
Index	BYTE	Index of axis in respective process
Process	BYTE	Related process
Unit	BYTE	Axis configuration unit (0 mm 1 inch)
ProgMode	BYTE	Programming mode (if moved by PLC)
Override	WORD	Override active on axis (if moved by PLC)
RealPosition	LREAL	Read position
IntpPositoin	LREAL	Interpolated position

Field	Type	Description
ProgPosition	LREAL	Programed position
HoldPosition	LREAL	Axis Hold position
RealFeed	LREAL	Read feed (unit/min)
IntpFeed	LREAL	Requested feed (unit/min)
FollowingErr	LREAL	Following error
DistToGo	LREAL	Distance at end of movement
EmgPosition	LREAL	Estimated position in emergency
TchdPosition	LREAL	Position probed
FiltPos	LREAL	Position calculated at exit from filters Feed calculated at exit from filters
PhisPoint	LREAL	Physical position requested of axis
PhisPos	LREAL	Physical position read by axis
FiltFeed	LREAL	Position calculated at exit from filters Feed calculated at exit from filters
Accel	LREAL	Instantaneous acceleration from interpolator
Jerk	LREAL	Instantaneous jerk from interpolator
Origin	WORD	Number of active origin on axis
OffsetNum	WORD	Number of active offset on axis
OffsetPrg	BYTE	Used mode to activate offset
TcpUprOffset	BYTE	Offset status with TCP or UPR
PrgStsOffset	WORD	Offset programming status
TotOffset	LREAL	Total Offset present on axis
ToolOffset	LREAL	Tool offset present on axis
OrigValue	LREAL	Offset due to origin
G92Offset	LREAL	Offset due to G92

Starting from OPENcontrol Rel. 3.0, the structure will have the following fields:

CartPos	LREAL	Cartesian position calculated
CartFeed	LREAL	Cartesian feed calculated
JointPos	LREAL	Joint feed calculated
JointFeed	LREAL	Joint feed read
RealJointPos	LREAL	Joint position read
RealCartPos	LREAL	Cartesian position read
RealIntpPos	LREAL	Interpolated position read

Starting from OPENcontrol Rel. 3.0.1, the structure will have the following fields:

AXDRTInfo1	LREAL	First real time drive information
AXDRTInfo2	LREAL	Second real time drive information
AXDRTInfo3	LREAL	Third real time drive information
AXDRTInfo4	LREAL	Fourth real time drive information

Starting from OPENcontrol Rel. 3.1, the structure will have the following fields:

AXDRTInfo5	LREAL	Fifth real-time drive information
AXDRTInfo6	LREAL	Sixth real-time drive information
AXDRTInfo7	LREAL	Seventh real-time drive information
AXDRTInfo8	LREAL	Eighth real-time drive information

Starting from OPENcontrol Rel. 3.2, the structure will have the following fields:

CamStatus	WORD	Electronic Cam status
CamMstId	BYTE	Master Id of the active cam (if the axis is interpolated)
CamActId	BYTE	Active action ID
IstFeed	LREAL	Instantenous feed read (unit/min)
ServoPos	LREAL	Servo position previous to the compensations

AxisSwSts	Mnemonic	Description
0x00000001	_AX_LOCKED	Axis in use
0x00000002	_AX_HOLD	Axis in hold
0x00000004	_AX_LOCKHOLD	Axis in use while in Hold
0x00000008	_AX_LOCKMSL	Axis used by Master/Slave
0x00000010	_AX_MM_INCH	0 mm 1 inch
0x00000040	_AX_PROCESS	Axis loaded from process
0x000000840	_AX_CAMMST	Master axis in one or more cams
0x00000100	_AX_MST	Master axis
0x00000200	_AX_SLV	Slave axis
0x00000800	_AX_CAMSLV	Slave axis in one or more cams
0x00001000	_AX_TCP	Axis in TCP use
0x00002000	_AX_UPR	Axis in UPR use
0x00004000	_AX_TCPLIN	Axis in TCP (linear) use
0x00008000	_AX_TCPROT	Axis in TCP (rotary) use
0x00010000	_SP_GEAR1	Gear 1 engaged
0x00020000	_SP_GEAR2	Gear 2 engaged
0x00040000	_SP_GEAR3	Gear 3 engaged
0x00080000	_SP_GEAR4	Gear 4 engaged
0x08000000	_AX_SWMLIMOPT	SW limits defined fro movements in manual ToolTip
0x10000000	_AX_SWLIMOPT	SW limits defined ToolTip
0x20000000	_AX_SWMLIMOP	SW limits defined for manual moves
0x40000000	_AX_SWLIMOP	Defined SW limits
0x80000000	_AX_MOVING	Axis being moved by interpolator

AxisHwSts	Mnemonic	Description
0x00000001	_DIS_NEGLOP	Negative SW operational limit disabled
0x00000002	_DIS_POSLOP	Positive SW operational limit disabled
0x00000010	_OVERTRAV_NEG	Negative over travel switch pressed
0x00000020	_OVERTRAV_POS	Positive over travel switch pressed
0x00000040	_HC_DONE	Homing done
0x00000080	_DEF_LOP	SW operational limits defined
0x00000100	_ZEROASS	On-going homing
0x00001000	_DIS_NEGLOPT	ToolTip negative SW operational limit disabled
0x00002000	_DIS_POSLOPT	ToolTip positive SW operational limit disabled

RealtSts	Mnemonic	Description
0x00000001	_IN_ERROR	Axis error
0x00000002	_TESTMODE	Axis in test mode

RealSts	Mnemonic	Description
0x00000004	_TOLL_HAP	Axis with tolerance executed
0x00000008	_IN_MOTION	Axis moving
0x00000010	_PROBE_DONE	Probing ended
0x00000020	_PROBE_FAULT	Probing error
0x00000040	_HOMING_ON	Homing ended
0x00000080	_MICRO_ON	Drive's micro-switch pressed
0x00000100	_POINT_UPD	RTM updated theoretical point
0x00004000	_MICRONEG	Homing negative direction for micro search
0x01000000	_DRIVE_READY1	Status Word (bit 14) AT
0x02000000	_DRIVE_READY2	Status Word (bit 15) AT
0x10000000	_CAM_ACTIVE	Electronic cam enabled on axis
0x00F00000	_MAINMODE_MSK	Mask for following 4 values (servo)
0x00000000	_MAINMODE_DIS	Axis disabled
0x00100000	_MAINMODE_TRQ	Torque driven axis
0x00200000	_MAINMODE_VEL	Velocity driven axis
0x00300000	_MAINMODE_POS	Position driven axis
0x000F0000	_SUB_MODE_MSK	Mask for next 4 values (loop)
0x00000000	_POS_MODE_SSO	Stand Still Open Loop
0x00010000	_POS_MODE_SCL	Space Closed Loop
0x00020000	_POS_MODE_VCL	Velocity Closed Loop
0x00030000	_POS_MODE_VOL	Velocity Open Loop

The _DRIVE_READY1 and _DRIVE_READY2 bits show the enabling status of a digital drive and must be read as a binary encoding having the values shown in the following table:

_DRIVE_READY2	_DRIVE_READY1	Meaning
0	0	Drive not ready for power up
0	1	Drive ready for main power on
1	0	Drive ready and main power applied
1	1	Drive ready to operate

SharingProc	Mnemonic	Description
0x80000000	_USE_SHARED	Shared axis
0x40000000	_USE_EXCLUSIVE	Shared axis in exclusive use
0x01000000	_USE_ByPLC	Shared axis used by PLC

MslStatus	Mnemonic	Description
0x0001	_AX_MSLUDASDA	UDA or SDA slave axis
0x0002	_AX_SYNCMSL	Master/Slave synchronisation
0x0004	_AX_SLVACTIVED	Enabled slave axis
0x0008	_AX_SYNCON	Axis synchronizing with the master
0x0010	_AX_UNSYNCON	Axis releasing the master
0x0020	_AX_XDAREADY	Slave axis with active XDA from CNC
0x0040	_AX_SLVREAL	Axis in realignment phase

PrbStatus	Mnemonic	Description
0x0001	_AX_PROBE_OK	Axis with OK probing
0x0002	_AX_PROBE_KO	Axis probing error
0x0004	_AX_PROBING	Axis probing
0x0008	_AX_PROBED_AV	Axis probing approaching

SpnStatus	Mnemonic	Description
0x0001	_SP_UPPER_RPM	Spindle speed exceeding the sup. SSL superior feed limit
0x0002	_SP_LOWER_RPM	Spindle speed lower than inf. SLL feed limit
0x0004	_SP_UPPER	Speed exceeding max gear speed
0x0008	_SP_ORIENT	Spindle orienting
0x0010	_SP_ORIENTED	Spindle oriented

AxType	Mnemonic	Description
0x0001	_AX_COORD	Coordinated axis (configuration)
0x0002	_AX_PTP	Point-to-point axis (configuration)
0x0004	_AX_ROT	Rotary axis
0x0008	_AX_DIG_GHOST	Inactive digital axis
0x0010	_AX_MANDRNOTR	Spindle axis without transducer
0x0020	_AX_MANDRTR	Spindle axis with transducer
0x0040	_AX_DIAM	Diametric axis
0x0100	_AX_VIRTUAL	Virtual axis
0x0200	_AX_OPTICAL	Axis with optical line
0x0800	_AX_MANDRRAMP	Spindle axis with ramps
0x1000	_AX_S_MASTER	Axis gantry master
0x2000	_AX_S_SLAVE	Axis gantry slave
0x4000	_AX_ROLLOVER	Axis rollover
0x8000	_AX_DIGITAL	Digital axis

OffsetPrg	Mnemonic	Description
0	_n_OFFSET	Offset not present
1	_h_OFFSET	Offset created by h programming
2	_t_OFFSET	Offset created by T programming
3	_u_OFFSET	Offset loaded otherwise

ProgrMode	Mnemonic	Description
0x01	_AX_ABSINC	Incremental programming
0x02	_AX_ROLOPT	Optimized rollover programming
0x04	_AX_EXTRAFEED	Excluded by feed calculation

TcpUprOffset	Mnemonic	Description
0x01	_TCPOffset	TCP in axis offsets
0x02	_UPROffset	UPR in axis offsets

PrgStsOffset	Mnemonic	Description
0x0001	_ABS_ORIGIN	Absolute origin presence
0x0002	_TMP_ORIGIN	Temporary origin presence

0x0004	_INC_ORIGIN	Incremental origin presence
0x0008	_MIRROR_MASK	Mirror presence
0x0010	_ROT_MASK	Rotation presence (in plane)
0x0040	_G92_ACTIVE	G92 presence
0x0080	_SCALE_FACT	Scale factor presence

CamStatus	Mnemonic	Description
0x0001	_AX_CAMRUN	Cam axis slave in execution
0x0002	_AX_CAMSTOP	Cam axis slave in stopping
0x0004	_AX_CAMEMERG	Cam axis slave in emergency
0x0008	_AX_CAMTOLLER	Cam axis slave in tolerance
0x0010	_AX_CAMWAIT	Cam axis slave in wait (after a reset or hold)

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160005	Too many axes required

Example:

```

Ret      : dword;
AxiInfo1 : AxisInfo_Struct;
AxiInfo2 : AxisInfo_Struct;

(* Reading axis status for axes ID 1 and 5 *)
Ret := GMC_ReadAxisInfo (1, AxiInfo1, 5, AxiInfo2) ;

```

3.60 GMC_ReadAxisInfoALL

GMC_ReadAxisInfoALL function reads the status of all axes.

Syntax

```
ret_code := GMC_ReadAxisInfoALL (Info, Offset) ;
```

Output parameters

<i>Info</i> (ARRAY OF AxisInfo_Struct)	Structure where reading information
<i>Offset</i> (INT)	Elements to skip when reading

Execution mode

Immediate

Use

“*Info*” parameter is an array of n elements, each of them being an AxisInfo_Struct type structure. According to the array dimension “n”, it will be possible to require information for axes with ID from 1 to n. For instance, defining a 10 elements array, it will be possible to read the information for axes with ID 1 to 10. If the array is higher than the maximum number of axes managed by the system, reading will be limited to the maximum number of axes that can be managed. All array axes require a synchronous axis status reading.

“*Offset*” parameter defines how many starting array elements have to be written. For instance, when writing 1 the first element array is not uploaded with axis data, all axes will be read from the second array element; of course, n-1 axes will be read.

Axis data are uploaded within an AxisInfo_Struct type structure. If required axis is missing, structure will be uploaded with 0 data; if the field Id within the structure is 0, which means the required axis does not exist. Data in the structure are:

Field	Type	Description
AxisSwSts	DWORD	Axis software status (Motion)
AxisHwSts	DWORD	Axis hardware status
RealtSts	DWORD	Axis real time status
SharingProc	DWORD	Processes sharing the axis
MslStatus	WORD	MasterSlave status
PrbStatus	WORD	Probing status
SpnStatus	WORD	Spindle activity status
VirtStatus	WORD	Virtualizations status
AxisType	WORD	Axis type
Clock	WORD	Axis clock in 0,1 ms
Name	BYTE	Axis name
Id	BYTE	Axis identifier
ActGear	BYTE	Active gear
MasterId	BYTE	Master axis id when the axis is Slave type
RollPitch	LREAL	Rollover pitch
Index	BYTE	Index of axis in respective process
Process	BYTE	Related process

Unit	BYTE	Axis configuration unit (0 mm 1 inch)
ProgrMode	BYTE	Programming mode (if moved by PLC)
Override	WORD	Override enabled on the axis (if moved by PLC)
RealPosition	LREAL	Read position
IntpPositoin	LREAL	Interpolated position
ProgPosition	LREAL	Programmed position
HoldPosition	LREAL	Axis in Hold position
RealFeed	LREAL	Read feed (unit/min)
IntpFeed	LREAL	Required feed (unit/min)
FollowingErr	LREAL	Following error
DistToGo	LREAL	Distance to go
EmgPosition	LREAL	Estimated emergency position
TchdPosition	LREAL	Probed position
FiltPos	LREAL	Position calculated at exit from filters
PhisPoint	LREAL	Real position required from the axis
PhisPos	LREAL	Real position read from the axis
FiltFeed	LREAL	Feed calculated at exit from filters
Accel	LREAL	Instant acceleration from interpolator
Jerk	LREAL	Instant jerk from interpolator
Origin	WORD	Origin number active on the axis
OffsetNum	WORD	Offset number active on the axis
OffsetPrg	BYTE	Offset programming mode
TcpUprOffset	BYTE	Offset status with TCP or UPR
PrgStsOffset	WORD	Offset programming status
TotOffset	LREAL	Total Offset on the axis
ToolOffset	LREAL	Tool offset on the axis
OrigValue	LREAL	Offset due to the origin
G92Offset	LREAL	Offset due to G92

Starting from OPENcontrol Rel. 3.0, the structure will have following fields too

CartPos	LREAL	Cartesian position calculated
CartFeed	LREAL	Cartesian feed calculated
JointPos	LREAL	Joint position calculated
JointFeed	LREAL	Joint feed calculated
RealJointPos	LREAL	Joint feed read
RealCartPos	LREAL	Cartesian position read
RealIntpPos	LREAL	Interpolated position read

Starting from OPENcontrol Rel. 3.0, the structure will have following fields too

AXDRTInfo1	LREAL	First real time drive information
AXDRTInfo2	LREAL	Second real time drive information
AXDRTInfo3	LREAL	Third real time drive information
AXDRTInfo4	LREAL	Fourth real time drive information

Starting from OPENcontrol Rel. 3.1, the structure will have the following fields too:

AXDRTInfo5	LREAL	Fifth real-time drive information
AXDRTInfo6	LREAL	Sixth real-time drive information
AXDRTInfo7	LREAL	Seventh real-time drive information
AXDRTInfo8	LREAL	Eighth real-time drive information

Starting from OPENcontrol Rel. 3.2, the structure will have the following fields too:

CamStatus	WORD	Electronic cam status
CamMstId	BYTE	Master Id (in case of interpolated axis) of the active cam
CamActId	BYTE	Id of the active action
IstFeed	LREAL	Instantaneous feed read (unit/min)
ServoPos	LREAL	Servo position before the compensations

AxisSwSts	Mnemonic	Description
0x00000001	_AX_LOCKED	Axis in use
0x00000002	_AX_HOLD	Axis in Hold
0x00000004	_AX_LOCKHOLD	Axis in use during Hold
0x00000008	_AX_LOCKMSL	Axis in use by Master/Slave
0x00000010	_AX_MM_INCH	0 mm 1 inch
0x00000040	_AX_PROCESS	Axis loaded from a process
0x00000100	_AX_MST	Master axis
0x00000200	_AX_SLV	Slave axis
0x00001000	_AX_TCP	Axis in TCP use
0x00002000	_AX_UPR	Axis in UPR use
0x00004000	_AX_TCPLIN	Axis in TCP use (linear)
0x00008000	_AX_TCPROT	Axis in TCP use (rotary)
0x00010000	_SP_GEAR1	Gear 1 inserted
0x00020000	_SP_GEAR2	Gear 2 inserted
0x00040000	_SP_GEAR3	Gear 3 inserted
0x00080000	_SP_GEAR4	Gear 4 inserted
0x08000000	_AX_SWMLIMOPT	ToolTip SW limits defined for manual motion
0x10000000	_AX_SWLIMOPT	ToolTip SW limits defined
0x20000000	_AX_SWMLIMOP	SW limits defined for manual motion
0x40000000	_AX_SWLIMOP	SW limits defined
0x80000000	_AX_MOVING	Axis being moved by interpolator

AxisHwSts	Mnemonic	Description
0x00000001	_DIS_NEGLOP	Negative SW operational limit disabled
0x00000002	_DIS_POSLOP	Positive SW operational limit disabled
0x00000010	_OVERTRAV_NEG	Negative over travel switch pressed
0x00000020	_OVERTRAV_POS	Positive over travel switch pressed
0x00000040	_HC_DONE	Homing done
0x00000080	_DEF_LOP	SW operational limits defined
0x00000100	_ZEROASS	on-going homing
0x00001000	_DIS_NEGLOPT	ToolTip negative SW operational limit disabled
0x00002000	_DIS_POSLOPT	ToolTip positive SW operational limit disabled

RealtSts	Mnemonic	Description
0x00000001	_IN_ERROR	Axis error
0x00000002	_TESTMODE	Axis in test mode
0x00000004	_TOLL_HAP	Axis with tolerance executed
0x00000008	_IN_MOTION	Axis moving
0x00000010	_PROBE_DONE	Probing ended
0x00000020	_PROBE_FAULT	Probing error
0x00000040	_HOMING_ON	Homing ended
0x00000080	_MICRO_ON	Micro on pressed drive
0x00000100	_POINT_UPD	RTM updated theoretical point
0x00004000	_MICRONEG	Homing negative direction for micro search
0x01000000	_DRIVE_READY1	Drive on
0x02000000	_DRIVE_READY2	Drive enabled
0x10000000	_CAM_ACTIVE	Electronic cam enabled on axis
0x00F00000	_MAINMODE_MSK	Mask for following 4 values (servo)
0x00000000	_MAINMODE_DIS	Axis disabled
0x00100000	_MAINMODE_TRQ	Torque driven axis
0x00200000	_MAINMODE_VEL	Velocity driven axis
0x00300000	_MAINMODE_POS	Position driven axis
0x000F0000	_SUB_MODE_MSK	Mask for next 4 values (loop)
0x00000000	_POS_MODE_SSO	Stand Still Open Loop
0x00010000	_POS_MODE_SCL	Space Closed Loop
0x00020000	_POS_MODE_VCL	Velocity Closed Loop
0x00030000	_POS_MODE_VOL	Velocity Open Loop

The _DRIVE_READY1 and _DRIVE_READY2 bits show the enabling status of a digital drive and must be read as a binary encoding having the values show in the The following table:

_DRIVE_READY2	_DRIVE_READY1	Meaning
0	0	Drive not ready for power up
0	1	Drive ready for main power on
1	0	Drive ready and main power applied
1	1	Drive ready to operate

SharingProc	Mnemonic	Description
0x80000000	_USE_SHARED	Shared axis
0x40000000	_USE_ECLUSIVE	Shared axis in exclusive use
0x01000000	_USE_ByPLC	Shared axis used by PLC

MslStatus	Mnemonic	Description
0x0001	_AX_MSLUDASDA	UDA or SDA slave axis
0x0002	_AX_SYNCMSL	Master/Slave synchronisation
0x0004	_AX_SLVACTIVED	Enabled slave axis
0x0008	_AX_SYNCON	Axis synchronizing with the master
0x0010	_AX_UNSYNCON	Axis releasing the master
0x0020	_AX_XDAREADY	Slave axis with active XDA from CNC
0x0040	_AX_SLVREAL	Axis in realignment stage

PrbStatus	Mnemonic	Description
0x0001	_AX_PROBE_OK	Axis with OK probing
0x0002	_AX_PROBE_KO	Axis probing error
0x0004	_AX_PROBING	Axis probing
0x0008	_AX_PROBED_AV	Axis probing approaching

SpnStatus	Mnemonic	Description
0x0001	_SP_UPPER_RPM	Spindle speed exceeding the sup. SSL superior feed limit
0x0002	_SP_LOWER_RPM	Spindle speed lower than inf. SLL feed limit
0x0004	_SP_UPPER	Speed exceeding max gear speed
0x0008	_SP_ORIENT	Spindle orienting
0x0010	_SP_ORIENTED	Spindle oriented

AxType	Mnemonic	Description
0x0001	_AX_COORD	Coordinated axis (configuration)
0x0002	_AX_PTP	Point-to-point axis (configuration)
0x0004	_AX_ROT	Rotary axis
0x0008	_AX_DIG_GHOST	Inactive digital axis
0x0010	_AX_MANDRNOTR	Spindle axis without transducer
0x0020	_AX_MANDRTR	Spindle axis with transducer
0x0040	_AX_DIAM	Diametric axis
0x0100	_AX_VIRTUAL	Virtual axis
0x0200	_AX_OPTICAL	Axis with optical line
0x0800	_AX_MANDRRAMP	Spindle axis with ramps
0x1000	_AX_S_MASTER	Axis gantry master
0x2000	_AX_S_SLAVE	Axis gantry slave
0x4000	_AX_ROLLOVER	Axis rollover
0x8000	_AX_DIGITAL	Digital axis

OffsetPrg	Mnemonic	Description
0	_n_OFFSETS	Offset not present
1	_h_OFFSETS	Offset created by h programming
2	_t_OFFSETS	Offset created by T programming
3	_u_OFFSETS	Offset loaded otherwise

ProgrMode	Mnemonic	Description
0x01	_AX_ABSINC	Incremental programing
0x02	_AX_ROLOPT	Optimized rollover programing
0x04	_AX_EXTRAFEED	Extra by feed calculation

TcpUprOffset	Mnemonic	Description
0x01	_TCPOffset	TCP in axis offsets
0x02	_UPROffset	UPR in axis offsets

PrgStsOffset	Mnemonic	Description
0x0001	_ABS_ORIGIN	Absolute origin presence
0x0002	_TMP_ORIGIN	Temporary origin presence

0x0004	_INC_ORIGIN	Incremental origin presence
0x0008	_MIRROR_MASK	Mirror presence
0x0010	_ROT_MASK	Rotation presence (in plane)
0x0040	_G92_ACTIVE	G92 presence
0x0080	_SCALE_FACT	Scale factor presence

CamStatus	Mnemonic	Description
0x0001	_AX_CAMRUN	Cam axis slave in execution
0x0002	_AX_CAMSTOP	Cam axis slave in stopping
0x0004	_AX_CAMEMERG	Cam axis slave in emergency
0x0008	_AX_CAMTOLLER	Cam axis slave in tolerance
0x0010	_AX_CAMWAIT	Cam axis slave in wait (after a reset or hold)

Return value

The following table lists all values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example

```

Ret      : dword;
AxiInfo : array [1..10] of AxisInfo_Struct;

(* Axis status reading for axes with ID from 1 to Id 10 *)
Ret := GMC_ReadAxisInfoALL (AxiInfo) ;

```

3.61 GMC_ChangePosLoop

Function GMC_ChangePosLoop changes axes servo loop mode.

Syntax:

```
ret_code := GMC_ChangePosLoop (ExecMode, ExecStatus, InterpId, Mode, AxisId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>Mode</i> (INT)	Servo loop mode
<i>AxisId</i> (INT)	Axis identifier [1...64]

Possible overload:

GMC_ChangePosLoop (INT,DWORD,INT,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

Defines the servo loop mode for up to 12 PLC axes. There are two servo loop modes (*Mode*):

Mode	Mnemonic	Description
2	SPACE_ERR	Error only interpolation
6	VFFSPC_ERR	Velocity and error interpolation

With following error motion mode, axis performs a smoother motion as the only offset to correct is the difference between the programmed motion and the real one: with this motion the axis “follows” the position required with a delay.

In addition to what aforementioned, with following error and feed motion mode, the control checks the axis keeps the feed required by the motion theoretical calculation. With this motion, axis is more “nervous” as it tries to meet the requests reducing the following error motion mode of the position required.

Servo loop mode is a parameter associated to a single axis (each axis has its value);servo loop mode is activated when the axis motion starts. Servo loop mode can't be modified while axis is moving.



A PLC interpolator has to be used to set the servo loop mode. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Undefined interpolator
0x00160031	Axis not related to the PLC
0x0016002C	Axes ID not accepted
0x0016002B	Axis ID not found
0x0016003A	Error mode
0x00160005	To many axes specified

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Sets servo error mode for axes ID 1 and 2 *)
Ret := GMC_ChangePosLoop (MODE_WAIT, UL0, InterpId, SPACE_ERR, 1, 2) ;

```

See also:

[GMC_ChangePosLoopALL](#)

3.62 GMC_ChangePosLoopALL

Function GMC_ChangePosLoopALL changes all axes servo loop mode.

Syntax

```
ret_code := GMC_ChangePosLoopALL (ExecMode, ExecStatus, InterId, Mode) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	Interpolator identifier PLC [1..40]
<i>Mode(INT)</i>	Servo loop mode

Execution mode:

Wait / NoWait

Use:

Defines the servo loop mode for all PLC axes. There are two servo loop modes (Mode):

Mode	Mnemonic	Description
2	SPACE_ERR	Error interpolation
6	VFFSPC_ERR	Velocity and error interpolation

With following error motion mode, axis performs a smoother motion as the only offset to correct is the difference between the programmed motion and the real one: with this motion the axis “follows” the position required with a delay.

In addition to what aforementioned, with following error and feed motion mode, the control checks the axis keeps the feed required by the motion theoretical calculation. With this motion, axis is more “nervous” as it tries to meet the requests reducing the following error motion mode of the position required.

Servo loop mode is a parameter associated to a single axis (each axis has its value); servo loop mode is activated when the axis motion starts. Servo loop mode can't be modified while axis is moving.



A PLC interpolator has to be used to set the right mode even if the operation is applied on all OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Undefined interpolator
0x0016003A	Error mode

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets VFF mode for all the PLC axes *)
Ret := GMC_ChangePosLoopALL (MODE_WAIT, UL0, InterpId, VFFSPC_ERR) ;
```

See also:

[GMC_ChangePosLoop](#)

3.63 GMC_SetAxisProperty

Function GMC_SetAxisProperty activates the axes setup mode.

Syntax:

```
ret_code := GMC_SetAxisProperty (ExecMode, ExecStatus, InterId, Mode, AxisId) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	Interpolator identifier PLC [1..40]
<i>Mode (BYTE)</i>	Programming mode
<i>AxisId (INT)</i>	Axis identifier [1...64]

Possible overload:

GMC_SetAxisProperty (INT,DWORD,INT,BYTE,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

Activates programming modes on up to 12 axes according to *Mode* parameter having the following values:

<i>Mode</i> =	0x01	MODE_INCR	Axis will be programmed in incremental mode. Values programmed are the movements following the last programmed position.
	0x02	MODE_ROLOPT	Improves rollover axes motion. If axis is programmed not in incremental mode, it reaches the programmed position using the shortest path.
	0x04	MODE_NOFEED	If axis is moved together with others, speed rate on profile doesn't consider the axis. In this case, programmed Feed is used to move other axes while the axis follows these moves. This performance is active only if the move instruction set as TRUE function VelOnlyLin.

From this moment on, each value programmed on the axis is influenced by the defined modalities, unless the programming mode is changed. Programming mode is a parameter associated to only one axis (each axis has a value) and in a motion there could be different axes programmed with different modalities.



A PLC interpolator has to be used to set the mode. Axes need not belong to the interpolator specified, but they may be all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Undefined interpolator
0x00160031	Axis not related to the PLC
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not found
0x00160005	Too many axes involved

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets incremental mode for axes ID 1 and 2 *)
Ret := GMC_SetAxisProperti (MODE_WAIT, ULO, InterpId, MODE_INCR, 1, 2) ;
```

See also:

[GMC_SetAxisPropertyALL](#)

3.64 GMC_SetAxisPropertyALL

Function `GMC_SetAxisPropertyALL` activates the axes setup mode for all the axes.

Syntax:

```
ret_code := GMC_SetAxisPropertyALL (ExecMode, ExecStatus, Interpld, Mode) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>Mode</i> (BYTE)	Programming mode

Execution mode:

Wait / NoWait

Use:

Enables for all the PLC axes some programming modes according to the *Mode* parameter. it may be:

Mode = 0x01 MODE_INCR	Incremental mode programming of the axis: programmed distances are the shifts towards the last position programmed.
0x02 MODE_ROLLOPT	Improves rollover axes motions: if axis isn't incrementally programmed, it reaches the programmed distance using the shorter path.
0x04 MODE_NOFEED	If axis is moved together with others, speed rate on profile doesn't consider the axis. In this case, programmed Feed is used to move other axes while the axis follows these moves. This performance is active only if the move instruction set as TRUE function VelOnlyLin.

From this moment on, each value programmed on the axis is influenced by the defined modalities, unless the programming mode is changed. Programming mode is a parameter associated to only one axis (each axis has a value) and in a motion there could be different axes programmed with different modalities.



A PLC interpolator has to be used to set the incremental mode even if the operation is applied on all the OPENcontrol axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Undefined interpolator

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets incremental mode for all the PLC axes *)
Ret := GMC_SetAxisPropertyALL (MODE_WAIT, UL0, InterpId, MODE_INCR) ;
```

See also:

[GMC_SetAxisProperty](#)

3.65 GMC_SetBitONposition

Function GMC_SetBitONposition sets a signal condition by axis position.

Syntax:

```
ret_code := GMC_SetBitONposition (ExecMode, ExecStatus.InterpId, Signal,  
                          AxisId, Position, Mode, Value) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>AxisId</i> (INT)	Axis identifier [1...64] of synchronism
<i>Position</i> (LREAL)	Synchronism position
<i>Mode</i> (INT)	Synchronism mode
<i>Value</i> (BOOL)	Value to be activated on signal

Output parameters:

<i>Signal</i> (BOOL)	Boolean variable to indicate synchronism
----------------------	--

Execution mode:

Wait / NoWait

Use:

This function is used to activate value “*Value*” on signal “*Signal*”, while waiting for the axis specified by “*AxisId*” to be in a given position as defined by “*Mode*” parameter; it may be:

Mode	Mnemonic	Description
10	CND_AXL	Sets Signal when axis AxisId is at an interpolated position lower than Position. The position must take into account the offsets active on the AxisId axis
11	CND_AXG	Sets Signal when axis AxisId is at an interpolated position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis
12	CND_AXLABS	Sets Signal when axis AxisId is at an interpolated position lower than Position. This position must be construed as absolute.
13	CND_AXGABS	Sets Signal when axis AxisId is at an interpolated position higher than or equal to Position. This position must be construed as absolute.
14	CND_AXLph	Sets Signal when axis AxisId is at a real position lower than Position. The position must take into account the offsets active on the AxisId axis.
15	CND_AXGph	Sets Signal when axis AxisId is at a real position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis.
16	CND_AXLABSph	Sets Signal when axis AxisId is at a real position lower than Position. This position must be construed as absolute.

Mode	Mnemonic	Description
17	CND_AXGABSpj	Sets Signal when axis AxisId is at a real position higher than or equal to Position. This position must be construed as absolute.



This function cannot be programmed from within the continuous stage.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016000D	Interpolation activities exhausted
0x00160036	Signal synchronism variable wrong
0x00160032	Synchronism condition wrong
0x0016005E	Rollover axis position wrong

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Indicates, on M0_1, when the position of the axis with ID 2 is higher than
200.0 *)
Ret := GMC_SetBitONposition (MODE_WAIT, UL0, InterpId, M0_1, 2,
                           200.0, CND_AXG, true) ;

```

See also:

[GMC_SetBitONvariable](#)

3.66 GMC_SetBitONvariable

Function `GMC_SetBitONvariable` sets a signal condition by value of a variable

Syntax:

```
ret_code := GMC_SetBitONvariable (ExecMode, ExecStatus, Interpld, Signal,
                                  Variable, VarValue, Mode, Value) ;
```

Input parameters

<code>ExecMode</code> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<code>ExecStatus</code> (DWORD)	Command execution status
<code>Interpld</code> (INT)	Interpolator identifier PLC [1..40]
<code>Variable</code> (INT)	Synchronism variable
<code>VarValue</code> (INT)	Synchronism value
<code>Mode</code> (INT)	Synchronism mode
<code>Value</code> (BOOL)	Value to be activated on signal

Output parameters

<code>Signal</code> (BOOL)	Boolean variable for indicating synchronism
----------------------------	---

Execution mode

Wait / NoWait

Use

This function is used to activate value “`Value`” on signal “`Signal`”, while waiting for the variable “`Variable`” to be in a given value as defined by “`Mode`” parameter; it may be:

<code>StopMode =</code>	40	<code>CND_VRLESS</code>	<i>Sets Value if value of Variable is lower than VarValue</i>
	41	<code>CND_VRGREAT</code>	<i>Sets Value if value of Variable is higher than VarValue</i>
	42	<code>CND_VREQUAL</code>	<i>Sets Value if value of Variable is equal to VarValue</i>
	43	<code>CND_VRDIF</code>	<i>Sets Value if value of Variable is different from VarValue</i>



This function cannot be programmed from within the continuous stage.

Return values:

<code>ret_code</code> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x0016000D	Interpolation activities exhausted
0x00160036	Signal synchronism variable wrong
0x00160032	Synchronism condition wrong
0x00160048	Variable synchronism variable wrong

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Shows, on M0_1, when value of variable MI300 becomes 200.0 *)
Ret := GMC_SetBitONvariable (MODE_WAIT, UL0, InterpId, M0_1, MI300,
                           200.0, CND_VEQUAL, true) ;
```

See also:

[GMC_SetBitONposition](#)

3.67 GMC_SetFeedOverride

Function GMC_SetFeedOverride determines the feed % for the axes specified.

Syntax:

```
ret_code := GMC_SetFeedOverride (ExecMode, ExecStatus, Interpld, AxisId, OverPerc) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>AxisId</i> (INT)	Axis identifier [1...64]
<i>OverPerc</i> (INT)	Percentage feed value in 0.01 %

Possible overload:

GMC_SetFeedOverride(INT,DWORD,INT,INT,INT[INT],...)

Execution mode:

Wait / NoWait

Use:

Defines the feed percentage to be applied to the axes specified (up to 12) associated with the PLC. In actual practice, if programmed feed is 10000 mm/min, by applying a percentage of 30% the maximum feed rate that an axis can reach will be 3000 mm/min. The admissible percentage rate is from 0% to 100%. Feed percentage is applied during movement execution, therefore it can be modified and implemented while a movement is underway.

Feed percentage is a parameter related to every axes (each axis has its own Feed Override value); when different axes are moved at the same time, interpolator uses the lower percentage value among the ones set on the axes involved.

Feed percentage is in cents therefore to obtain a 30% percentage the value to set is $30 \times 100 = 3000$.



It is necessary to rely on a PLC interpolator to be able to set the override %; however, the axes need not necessarily belong to the interpolator specified, and they may include all the OPENcontrol system axes associated with the PLC.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not defined
0x00160031	Axis not associated with PLC
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x00160005	Too many axes specified

Example

```
Ret      : dword;
InterpId : INT;
...
(* Sets % to 0 for axes with ID 1 and 2 *)
Ret := GMC_SetFeedOverride (MODE_WAIT, UL0, InterpId, 0, 1, 2) ;
```

See also

[GMC_SetFeedOverrideALL](#)

3.68 GMC_SetFeedOverrideALL

Function GMC_SetFeedOverrideALL modifies the feed % for the axes specified.

Syntax:

```
ret_code := GMC_SetFeedOverrideALL (ExecMode, ExecStatus, InterpId, OverPerc) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>OverPerc</i> (INT)	Percentage feed value in 0.01 %

Execution mode

Wait / NoWait

Use

Defines the feed percentage to be applied to the axes specified (up to 12) associated with the PLC. In actual practice, if programmed feed is 10000 mm/min, by applying a percentage of 30% the maximum feed rate that an axis can reach will be 3000 mm/min. The admissible percentage rate is from 0% to 100%.

Feed percentage is applied during movement execution and therefore it can be modified, and implemented, while a movement is underway.

The feed percentage parameter is associated with the individual axes (each axis has a specific feed percentage of its own to be applied); when several axes are moved during the same movement, the percentage value used is the lowest between those set for the various axes involved in the movement.

The feed percentage is expressed in hundredths and therefore to request a percentage of 30% we must set a value of $30 \times 100 = 3000$.

 It is necessary to rely on a PLC interpolator to be able to set the override %, even if the operation is applied to all the OPENcontrol system axes currently associated with the PLC.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not defined

Example

```
Ret      : dword;
InterpId : INT;
...
(* Sets override % to 80% for all the axes of the PLC *)
Ret := GMC_SetFeedOverrideALL (MODE_WAIT, ULO, InterpId, 8000) ;
```

See also

GMC_SetFeedOverride

3.69 GMC_SetFeed

The GMC_SetFeed function sets the feedrate on the profile.

Syntax

```
ret_code := GMC_SetFeed (ExecMode, ExecStatus, InterpId, RapMode, Feed) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>RapMode</i> (INT)	Defines whether use Rapid or working mode
<i>Feed</i> (LREAL)	Feed value in unit/min

Execution mode

Wait / NoWait

Use

Sets the feed used in following motions (unless motion itself has the feed as a parameter). Feed set is limited to the interpolator, therefore it is possible too have several active Feed, but on different interpolators. Feed is measured unit/min. Feed means vector feed, that is the feedrate of all axes moved and not the single ones.

The *RapMode* parameter allows to configure whether the dynamic referred to movements must respect the features of rapid and working motions; according to this, acceleration, jerk and minimum ramp time parameters for rapid or working dynamics will be used.

RapMode	Mnemonic	Description
0	MODE_RAPID	Motions will use the rapid motion dynamics.
1	MODE_WORK	Motions will use the working movement dynamics.

Return values

The following table lists all possible values *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0016004B	Wrong dynamic RapMode mode
0x00160014	Undefined interpolator

Example

```
Ret      : dword;
InterpId : INT;
...
(* Sets a mm/min 10000.0 Feed in rapid *)
Ret := GMC_SetFeed (MODE_WAIT, ULO, InterpId, MODE_RAPID, 10000.0) ;
```

See also

GMC_Move, GMC_MoveOnPosition, GMC_MoveOnSignal, GMC_MoveRoundOFF, GMC_MoveUntilSignal, GMC_MoveUntilVariable

3.70 GMC_SetJerk

This function defines the S ramps JRK.

Syntax:

```
ret_code := GMC_SetJerk (ExecMode, ExecStatus, InterpId, JRK) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>JRK</i> (LREAL)	JRK value

Execution mode:

Wait / NoWait

Use:

Defines the Jerk value to be adopted for S ramps. A value of Jerk of 1.0 determines an acceleration, or deceleration, time corresponding to 1.45833 times the acceleration or deceleration time applied when a linear ramp is used, and ensures that the maximum acceleration value requested during a ramp does not exceed the programmed value. Jerk values higher than 1.0 cause the time to increase proportionately to the value programmed with this function; in this case, the maximum acceleration value requested will be lower than the value programmed. It is not possible to exceed a value of 50.0.

Jerk values lower than 1.0 cause the time to decrease, while, at the same time, the maximum acceleration value requested will exceed the value programmed. It is not possible to go below a value of 0.5.

The jerk value set is local to the interpolator in which it is set and hence we may have several different JRK values active at the same time, on different interpolators.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160047	Wrong JRK value
0x00160014	Undefined interpolator

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets a JRK=2.0 *)
Ret := GMC_SetJerk (MODE_WAIT, UL0, InterpId, 2.0) ;
```

See also:

GMC_SetRamp, GMC_SetRampALL

3.71 GMC_SetRamp

Function GMC_SetRamp changes the ramp mode for the axes

Syntax:

```
ret_code := GMC_SetRamp (ExecMode, ExecStatus, InterpId, Mode, AxisId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>RampMode</i> (INT)	Ramp mode
<i>AxisId</i> (INT)	Axis identifier [1...64]

Possible overload:

GMC_SetRamp(INT,DWORD,INT,INT,INT[,INT]...)

Execution mode:

Wait / NoWait

Use:

Defines the ramp mode for one or more axes (up to 12) associated with the PLC. Four different ramp modes can be defined by means of the “RampMode” parameter:

RampMode =	1	LINEAR_RAMP	Linear ramp
	2	S_RAMP	S ramp
	3	TRAPEZ_RAMP	Trapezoidal ramp
	4	JRK_RAMP	Jerk limitation ramp

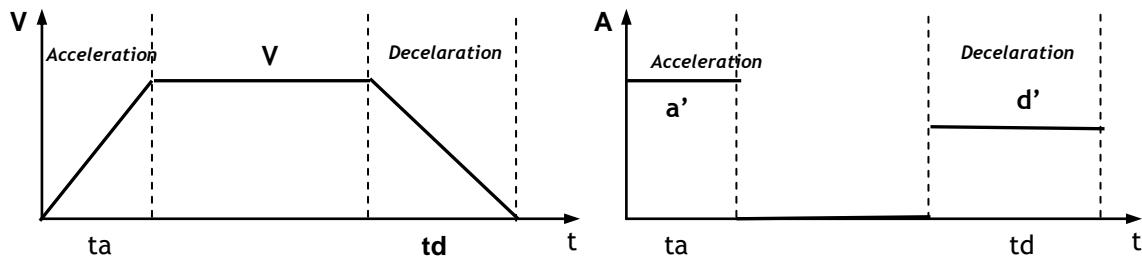
Ramp mode is a parameter that is associated with a single axis (each axis has a value of its own); when several axes are involved in a movement, if all the axes have activated the same ramp mode, that mode is used, otherwise the “gentlest” ramp is used.



It is necessary to rely on a PLC interpolator to be able to set the ramp mode; however, the axes need not necessarily belong to the interpolator specified, and they may include all the PENcontrolsyste axes currently associated with the PLC.

Linear ramp

The linear ramp is the simplest of all, at the acceleration or deceleration stage, speed increases or decreases (per time unit) by a constant value corresponding to the acceleration or deceleration configured for the axis. Speed and acceleration curves (with different acceleration and deceleration values) are shown in the figure below.

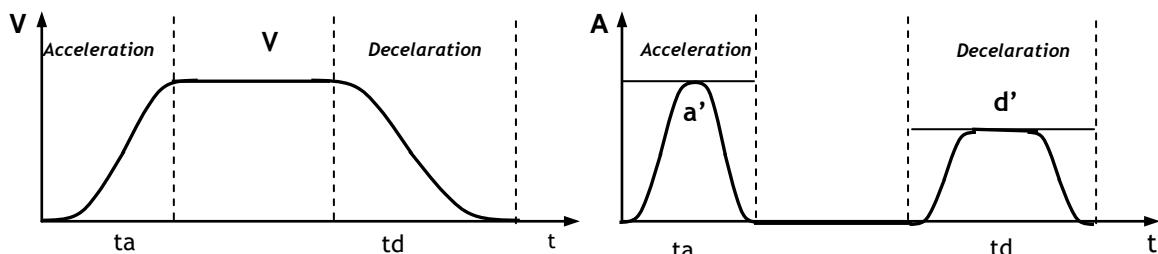


Ramp time (ta , td) is also affected by the “minimum ramp time” parameter configured for each axis. If the time calculated for a ramp does not meet the minimum ramp time requirement, i.e., is lower, the ramp will be recalculated so as to comply with this requirement.

S ramp

An S ramp describes the acceleration or deceleration stage through a polynomial curve having the characteristic of varying continuously the acceleration requested of the axis; moreover it allows to manage, through a parameter called JRK, the maximum acceleration reached in the central portion of the ramp, and the time it takes to execute the ramp.

Assuming that the value of the JRK parameter is 1 (predetermined value), the speed curves will be as follows:



In this case, the acceleration and deceleration times will be the same as the times associated with a linear ramp multiplied by a factor of 1.45833. The acceleration value reached in the central stage of the ramp will correspond to the acceleration configured. By changing the JRK value (see function `GMC_SetJerk`), we can decrease (or increase) the acceleration/deceleration time. By decreasing JRK we decrease the time it takes to execute the ramp; by increasing it, we increase the time it takes to execute the ramp. The variation in execution time is proportional to the value of JRK; thus, if we set JRK to 2, we shall double the ramp execution time, if we set it to 0.5, we shall halve it.

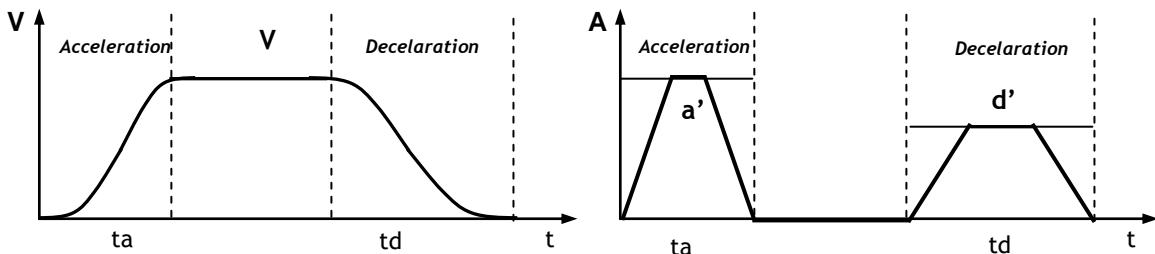
However, the moment we decrease JRK, and hence the acceleration time decreases, we increase the maximum acceleration value reached in the central part of the ramp, which therefore will exceed the acceleration value configured. The same happens if we increase JRK: in this case, the maximum acceleration reached will be lower than the set value.

Ramp time (ta , td) is also affected by the “minimum ramp time” parameter configured for each axis. If the time calculated for a ramp does not meet the minimum ramp time requirement, i.e., is lower, the ramp will be recalculated so as to comply with this requirement.

N.B. The minimum ramp time is the time it takes to reach the maximum acceleration (Jerk application time) and hence total ramp time is twice the minimum time.

Trapezoidal ramp

A trapezoidal ramp describes the acceleration or deceleration time by way of a trapezium that has the characteristic of applying a non-constant acceleration to the axis; moreover, it makes it possible to manage, through the jerk configuration parameter, the maximum acceleration reached in the central part of the ramp and the time it takes to execute the ramp itself.



Ramp time (ta, td) is also affected by the “minimum ramp time” parameter configured for each axis. If the time calculated for a ramp does not meet the minimum ramp time requirement, i.e., is lower, the ramp will be recalculated so as to comply with this requirement.

N.B. The minimum ramp time is the time it takes to reach the maximum acceleration (Jerk application time) and hence total ramp time is more than twice the minimum time (or twice the minimum time in the absence of a constant acceleration segment).

Ramp with Jerk Limitation

A ramp with Jerk Limitation is a development of the S ramp. With the S ramp we try to give continuity to acceleration variations without considering how this variation is executed, by introducing the jerk concept, instead, we determine how to vary the acceleration over time and to what extent. In actual fact, Jerk is the derivative before the acceleration (i.e., it determines how the acceleration varies over time; its unit of measure in fact is mm/sec³) and it may be likened to what the acceleration is for feed rate.

Ramps with Jerk Limitation are based on the S ramps, and therefore start from the latter, but they prevent the acceleration from changing over time by a quantity greater than the Jerk configured for the axis. Ultimately, the acceleration values set for an axis being the same, a ramp Jerk Limitation CANNOT be more effective than an S ramp, it can only make things worse.

The advantage we can get from ramps with Jerk Limitation lies in the fact that, since the acceleration variation is kept under control, it is possible to raise to the maximum the accelerations that an axis can withstand, and thereby obtain superior results in terms of time, quality of the motion, motor wear.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Undefined interpolator
0x00160031	Axis not associated to the PLC
0x0016002C	Unaccepted axis id
0x0016002B	Axis ID not found
0x0016003B	Wrong ramp
0x00160005	Too many axes

Example:

```
Ret      : dword;
InterpId : INT;
...
(*Sets an S ramp for axes with ID 1 and ID 2 *)
Ret := GMC_SetRamp (MODE_WAIT, ULO, InterpId, S_RAMP, 1, 2) ;
```

See also:

[GMC_SetRampALL](#), [GMC_SetJerk](#)

3.72 GMC_SetRampALL

Function GMC_SetRampALL changes ramp mode for all the axes.

Syntax:

```
ret_code := GMC_SetRampALL (ExecMode, ExecStatus, InterId, RampMode) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	PLC interpolator identifier [1..40]
<i>RampMode (INT)</i>	Ramp mode

Execution mode:

Wait / NoWait

Use:

Defines the ramp mode for one or more PLC axes (max 12). “RampMode” parameters refers to the following 4 ramp modes :

RampMode =	1 LINEAR_RAMP	Linear ramp
	2 S_RAMP	S ramp
	3 TRAPEZ_RAMP	Trapezoidal ramp
	4 JRK_RAMP	Jerk limitation ramp

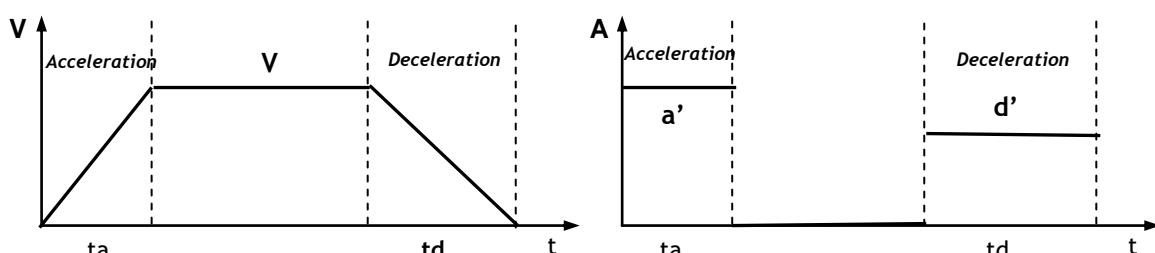
Ramp mode is a parameter that is associated with a single axis (each axis has a value of its own); when several axes are involved in a movement, if all the axes have activated the same ramp mode, that mode is used, otherwise the “gentlest” ramp is used.



It is necessary to rely on a PLC interpolator to be able to set the ramp mode; however, the axes need not necessarily belong to the interpolator specified, and they may include all the PENcontrols system axes currently associated with the PLC.

Linear ramp

The linear ramp is the simplest of all, at the acceleration or deceleration stage, speed increases or decreases (per time unit) by a constant value corresponding to the acceleration or deceleration configured for the axis. Speed and acceleration curves (with different acceleration and deceleration values) are shown in the figure below.

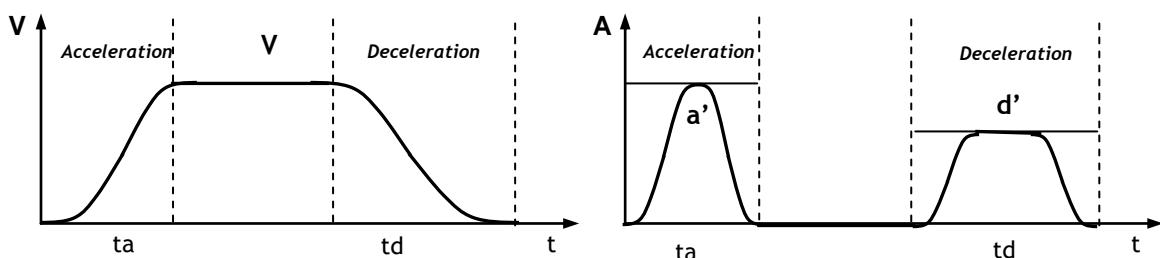


Ramp time (ta , td) is also affected by the “minimum ramp time” parameter configured for each axis. If the time calculated for a ramp does not meet the minimum ramp time requirement, i.e., is lower, the ramp will be recalculated so as to comply with this requirement.

S ramp

An S ramp describes the acceleration or deceleration stage through a polynomial curve having the characteristic of varying continuously the acceleration requested of the axis; moreover it allows to manage, through a parameter called JRK, the maximum acceleration reached in the central portion of the ramp, and the time it takes to execute the ramp.

Assuming that the value of the JRK parameter is 1 (predetermined value), the speed curves will be as follows:



In this case, the acceleration and deceleration times will be the same as the times associated with a linear ramp multiplied by a factor of 1.45833. The acceleration value reached in the central stage of the ramp will correspond to the acceleration configured. By changing the JRK value (see function GMC_SetJerk), we can decrease (or increase) the acceleration/deceleration time. By decreasing JRK we decrease the time it takes to execute the ramp; by increasing it, we increase the time it takes to execute the ramp. The variation in execution time is proportional to the value of JRK; thus, if we set JRK to 2, we shall double the ramp execution time, if we set it to 0.5, we shall halve it.

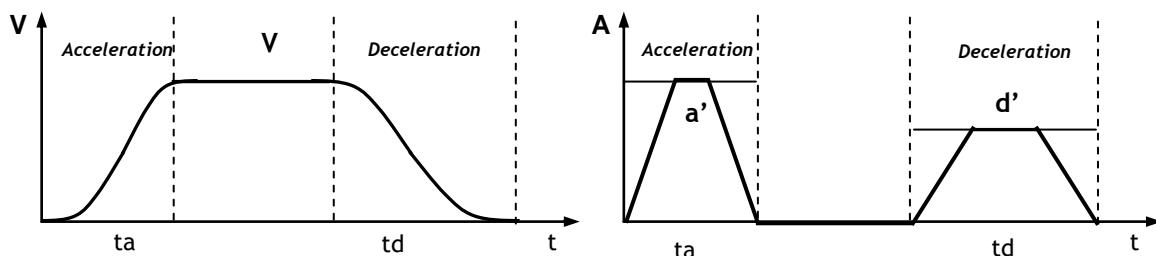
However, the moment we decrease JRK, and hence the acceleration time decreases, we increase the maximum acceleration value reached in the central part of the ramp, which therefore will exceed the acceleration value configured. The same happens if we increase JRK: in this case, the maximum acceleration reached will be lower than the set value.

Ramp time (ta , td) is also affected by the “minimum ramp time” parameter configured for each axis. If the time calculated for a ramp does not meet the minimum ramp time requirement, i.e., is lower, the ramp will be recalculated so as to comply with this requirement.

N.B. The minimum ramp time is the time it takes to reach the maximum acceleration (Jerk application time) and hence total ramp time is twice the minimum time.

Trapezoidal ramp

A trapezoidal ramp describes the acceleration or deceleration time by way of a trapezium that has the characteristic of applying a non-constant acceleration to the axis; moreover, it makes it possible to manage, through the jerk configuration parameter, the maximum acceleration reached in the central part of the ramp and the time it takes to execute the ramp itself.



Ramp time (ta , td) is also affected by the “minimum ramp time” parameter configured for each axis. If the time calculated for a ramp does not meet the minimum ramp time requirement, i.e., is lower, the ramp will be recalculated so as to comply with this requirement.

N.B. The minimum ramp time is the time it takes to reach the maximum acceleration (Jerk application time) and hence total ramp time is more than twice the minimum time (or twice the minimum time in the absence of a constant acceleration segment).

Ramp with Jerk Limitation

A ramp with Jerk Limitation is a development of the S ramp. With the S ramp we try to give continuity to acceleration variations without considering how this variation is executed, by introducing the jerk concept, instead, we determine how to vary the acceleration over time and to what extent. In actual fact, Jerk is the derivative before the acceleration (i.e., it determines how the acceleration varies over time; its unit of measure in fact is mm/sec³) and it may be likened to what the acceleration is for feed rate.

Ramps with Jerk Limitation are based on the S ramps, and therefore start from the latter, but they prevent the acceleration from changing over time by a quantity greater than the Jerk configured for the axis. Ultimately, the acceleration values set for an axis being the same, a ramp Jerk Limitation CANNOT be more effective than an S ramp, it can only make things worse.

The advantage we can get from ramps with Jerk Limitation lies in the fact that, since the acceleration variation is kept under control, it is possible to raise to the maximum the accelerations that an axis can withstand, and thereby obtain superior results in terms of time, quality of the motion, motor wear.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Undefined interpolator
0x0016003B	Wrong ramp mode

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets trapezoidal ramp for all the PLC axes *)
Ret := GMC_SetRampALL (MODE_WAIT, UL0, InterpId, TRAPEZ_RAMP) ;
```

See also:

GMC_SetRamp, GMC_SetJerk

3.73 GMC_SetOrigin

Function GMC_SetOrigin configures an origin on the axes.

Syntax:

```
ret_code := GMC_SetOrigin (ExecMode, ExecStatus, InterId, Origin, AxisId, Value) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	Interpolator identifier PLC [1..40]
<i>Origin (INT)</i>	Origin number
<i>AxisId (INT)</i>	Axis Id [1...64] for up to 12 axes
<i>Value (LREAL)</i>	Origin value

Possible overload:

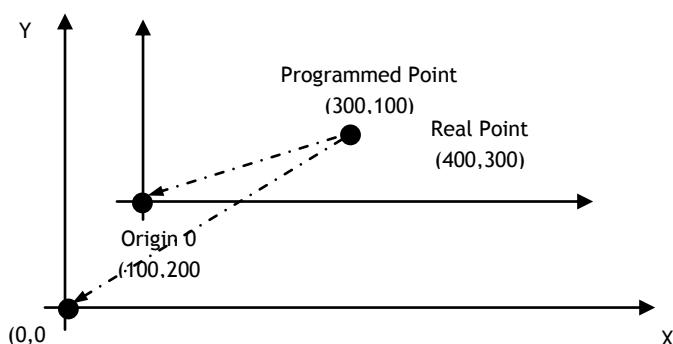
GMC_SetOrigin(INT,DWORD,INT,INT,INT,LREAL[,INT,LREAL]...)

Execution mode:

Wait / NoWait

Use:

For each axis being used by the PLC, it is possible to manage 32 private origins (numbered from 0 to 31). Through this function we configure, for a certain number of axes (up to 12), the value to be assigned to the origin. Once the origin has been activated (the activation operation is by means of function GMC_ActivateOrigin), all the points programmed for a given axis will be referred to this new origin. For example, once we have configured origin 0 for the X axis with a value of 100 and have activated this origin, 100 must be added to each point programmed (if the value programmed is 300, in actual fact the axis will move to 400).



It is possible to make different configuration calls for the same origin. An axis maintains its configuration for an origin until the same origin is configured again on the axis. To reset the origin, set it to 0.



It is necessary to rely on a PLC interpolator to be able to set an origin for the axes; however, the axes need not necessarily belong to the interpolator specified, they may include all the OPENcontrol system axes currently associated with the PLC.

An Origin becomes active only when the origin activation command, i.e., GMC_ActivateOrigin, is given.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160005	Too many axes involved
0x00160050	Wrong origin number
0x0016002C	Unaccepted ID axis
0x0016002B	ID axis not found
0x00160031	Axis not associated with PLC

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets origin 1 for axis with ID 1 value 100.0 for axis with ID 2 value 77.0*)
Ret := GMC_SetOrigin (MODE_WAIT, UL0, InterpId, 1, 1, 100.0, 2, 77.0) ;
```

See also:

[GMC_ActivateOrigin](#)

3.74 GMC_ActivateOrigin

Function GMC_ActivateOrigin activates an origin.

Syntax:

```
ret_code := GMC_ActivateOrigin (ExecMode, ExecStatus, InterId, Origin) ;
```

Input parameters:

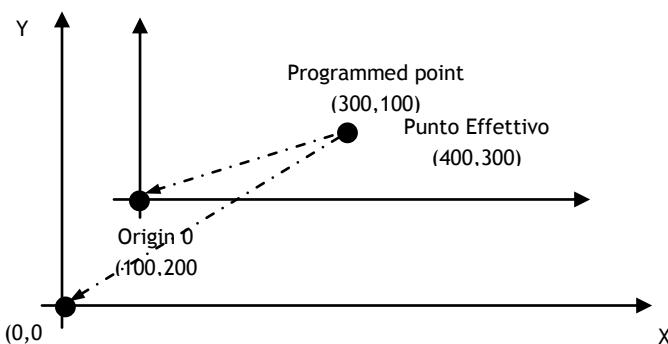
<i>ecMode</i> (INT)	Execution mode command [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Execution status command
<i>InterId</i> (INT)	PLC interpolator identifier [1..40]
<i>Origin</i> (INT)	Origin number

Execution mode:

Wait / NoWait

Use:

For each axis being used by the PLC, it is possible to manage 32 private origins (numbered from 0 to 31). Through this function we configure, for a certain number of axes with an origin already configured, the value to be assigned to the origin. Once the origin has been activated all the points programmed for a given axis will be referred to this new origin. For example, once we have configured origin 0 for the X axis with a value of 100 and have activated this origin, 100 must be added to each point programmed (if the value programmed is 300, in actual fact the axis will move to 400).



It is necessary to rely on a PLC interpolator to be able to set an origin for the axes Origin is activated on all the PLC axes.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160050	Wrong origin number
0x00160051	No axis associated to the origin requested

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Activates origin 1 *)
Ret := GMC_ActivateOrigin (MODE_WAIT, ULO, InterpId, 1) ;
```

See also:

[GMC_SetOrigin](#)

3.75 GMC_SetResetQuote

Function GMC_SetResetQuote configures the axis position reset.

Syntax:

```
ret_code := GMC_SetResetQuote (ExecMode, ExecStatus, Interpld, AxisId, RollStep, Offset) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>Interpld (INT)</i>	Interpolator identifier PLC [1..40]
<i>AxisId (INT)</i>	Axis Identifier [1...64]
<i>RollStep (LREAL)</i>	Reset rollover step
<i>Offset (LREAL)</i>	Offset to associate with reset

Execution mode:

Wait / NoWait

Use:

Through this function we configure, for the axis concerned, the parameters to be used when the “axis position reset” command is executed, to be executed, unconditionally, upon the execution of function **GMC_ActivateResetQuote**.

Through the “axis position reset” operation we execute the reset of the axis position, i.e., upon the execution of the GMC_ActivateResetQuote function, the axis moves to position 0. This operation is similar to the execution of an origin, but, unlike the latter, it is applied “on the go”; the system calculates its own internal origin, referred to as “reset origin”.

By means of the “Offset” parameter we may define an offset to be associated with an axis during the reset operation. For example, if the “Offset” parameter is set to 50.0, when the position reset operation is activated, the axis will move to position 50.0 instead of 0.0.

By means of the “RollStep” parameter we may define the modulus to be applied during the axis position reset. For instance, if a rotary axis is at 3610 degrees and we execute a GMC_SetResetQuote function with a “RollStep” of 360, upon the execution of the GMC_ActivateResetQuote function, the axis will move to a position of 10 degrees instead of 0.0.

The formula used to apply the LRQ is as follows:

New axis position = Current position RollStep mod+ Offset (if RollStep is other than 0)

New axis position = Offset (if RollStep is equal to 0)



It is necessary to rely on a PLC interpolator to be able to set the axis reset position; however, the axis needs not necessarily belong to the interpolator specified, it may be one of OPENcontrol system axes associated with the PLC.

A position reset becomes active only when the position reset command, i.e., GMC_ActivateResetQuote, is given.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not found
0x00160031	Axis not associated to the PLC

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets axis ID 1 reset quote with 140.0 value *)
Ret := GMC_SetResetQuote (MODE_WAIT, UL0, InterpId, 1, 0.0, 140.0) ;
```

See also:

[GMC_ActivateResetQuote](#), [GMC_SetResetQuoteOnPosition](#), [GMC_SetResetQuoteOnSignal](#),
[GMC_SetResetQuoteOnMarker](#), [GMC_SetResetQuoteShift](#).

3.76 GMC_ActivateResetQuote

Function GMC_ActivateResetQuote resets an axis.

Syntax:

```
ret_code := GMC_ActivateResetQuote (ExecMode, ExecStatus, Interpld, EndSignal, AxisId) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Execution mode command [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Execution status command
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>AxisId</i> (INT)	Axis identifier [1...64]

Output parameters:

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (signal set to TRUE)
-------------------------	--

Possible overload:

GMC_ActivateResetQuote (INT,DWORD,INT,BOOL,INT[,INT]..)

Execution mode:

Wait / NoWait

Use:

It performs the “axis point reset” function previously configured by means of functions GMC_SetResetQuote, GMC_SetResetQuoteONPosition, GMC_SetResetQuoteONSignal, GMC_SetResetQuoteShift and GMC_SetResetQuoteONMarker.

At the end of the execution, the “EndSignal” signal is set to true. Parameter “AxisId” may be duplicated so as to request the axis point reset operation for several axes, up to 12. This function may be executed both with the axis stationary and with the axis moving; in the latter case, needless to say, the GMC_ActivateResetQuote function cannot be executed by the interpolator which is moving the axis.

Through the “axis point reset” function the axis point is reset, i.e., the axis position becomes 0. It is similar to the execution of an origin, but, unlike the latter, it is applied “in progress”; the system will calculate an internal origin referred to as “reset origin” .

The “axis point reset” configuration function makes available two parameters, which will be used during the execution of the GMC_ActivateResetQuote function.

By means of the “Offset” parameter we may define an offset value to be associated with an axis during a reset. For instance, if the “Offset” parameter is set to 50.0, when the point reset operation is activated, the position of the axis will be 50.0 in lieu of 0.0.

The “RollStep” parameter makes it possible to define the modulus to be applied during the axis point reset. For example, if the position of a rotary axis is 3610 degrees and a reset request is executed with a 360 “RollStep”, upon the execution of the GMC_ActivateResetQuote function, the position of the axis will be 10 degrees instead of 0.0 degrees.

The formula set for the application of GMC_ActivateResetQuote is :

New axis position = Current position mod RollStep + Offset	(if RollStep is not 0)
New axis position = Offset	(if RollStep is 0)



If the operation follows a GMC_SetResetQuoteONMarker request, a veritable axis reset operation is executed. In this case the axis must not be moving. A PLC interpolator has to be used to execute the axis point reset function; the axis need not belong to the interpolator specified: it may be one of the OPENcontrol system axes currently associated with the PLC.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator undefined
0x00160003	Too many activities running
0x00160004	Move elements finished
0x00160005	Too many axes involved
0x0016002C	Unaccepted axis ID
0x0016002B	Axis ID not found
0x00160031	Axis not related to the PLC
0x00160028	Axis in use reset on marker not permitted
0x00160030	Axis in use as slave reset on marker not permitted

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Executes axes ID 1 and ID 2 homing*)
Ret := GMC_ActivateResetQuote (MODE_WAIT, ULO, InterpId, M0_1, 1, 2) ;
```

See also:

[GMC_SetResetQuote](#), [GMC_SetResetQuoteOnPosition](#), [GMC_SetResetQuoteOnSignal](#),
[GMC_SetResetQuoteOnMarker](#), [GMC_SetResetQuoteShift](#).

3.77 GMC_SetResetQuoteONmarker

Function **GMC_SetResetQuoteONmarker** configures the axis reset position according to the marker.

Syntax:

```
ret_code := GMC_SetResetQuoteONmarker (ExecMode, ExecStatus, InterpId, AxisId, Offset) ;
```

Input parameters:

<i>ecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterpId (INT)</i>	PLC interpolator identifier [1..40]
<i>AxisId (INT)</i>	Axis identifier [1...64]
<i>Offset (LREAL)</i>	Offset to associate with reset

Execution mode:

Wait / NoWait

Use:

Through this function we configure, for the axis concerned, the parameters to be used when the “axis position reset” command is executed, to be executed, **immediately**, upon the execution of function **GMC_ActivateResetQuote**. Through the “axis position reset” operation we execute the reset of the axis position, i.e., upon the execution of the **GMC_ActivateResetQuote** function, the axis moves to position 0 according to the marker position. The system will “physically” executes the axis position reset. By means of the “Offset” parameter we may define an offset to be associated with an axis during the reset operation. For example, if the “Offset” parameter is set to 50.0, when the position reset operation is activated, the axis will move to position 50.0 instead of 0.0.



It is necessary to rely on a PLC interpolator to be able to set the axis reset position; however, the axis needs not necessarily belong to the interpolator specified, it may be one of OPENcontrol system axes associated with the PLC. A position reset becomes active only when the position reset command, i.e., **GMC_ActivateResetQuote**, is given..

Return values:

<i>ret_code (HEX)</i>	Description
0x00000000	Function executed without errors
0x0016002C	ID axis unaccepted
0x0016002B	Axis ID not found
0x00160031	Axis not associated to PLC

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets reset quote for axis ID 1 being 20.0 *)
Ret := GMC_SetResetQuoteONmarker (MODE_WAIT, UL0, InterpId, 1, 20.0) ;
```

See also: **GMC_ActivateResetQuote**, **GMC_SetResetQuoteOnPosition**, **GMC_SetResetQuoteOnSignal**, **GMC_SetResetQuote**, **GMC_SetResetQuoteShift**.

3.78 GMC_SetResetQuoteONPosition

Function GMC_SetResetQuoteONposition configures the axis reset point as a function of axis position.

Syntax:

```
ret_code := GMC_SetResetQuoteONposition (ExecMode, ExecStatus, Interpld, AxisId, Position,  
                  Mode, RollStep, Offset) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	PLC interpolator identifier [1..40]
<i>AxisId</i> (INT)	Axis identifier [1...64]
<i>Position</i> (LREAL)	Reset position
<i>Mode</i> (INT)	Reset mode
<i>RollStep</i> (LREAL)	Reset rollover step
<i>Offset</i> (LREAL)	Offset to be associated with reset

Execution mode:

Wait / NoWait

Use:

Through this function we configure, for the axis concerned, the parameters to be used when the “axis position reset” command is executed, to be executed, **depends on axis position**, upon the execution of function GMC_ActivateResetQuote. Through the “axis position reset” operation we execute the reset of the axis position, i.e., upon the execution of the GMC_ActivateResetQuote function, the axis moves to position 0. This operation is similar to the execution of an origin, but, unlike the latter, it is applied “on the go”; the system calculates its own internal origin, referred to as “reset origin”.

By means of the “Mode” and “Position” parameters we may define when the reset operation is activated:

Mode	Mnemonic	Description
10	CND_AXL	Reset when the axis <i>StartAxisId</i> is at an interpolated position lower than <i>Position</i> . The position will consider active offset on <i>StartAxisId</i>
11	CND_AXG	Reset when the axis <i>StartAxisId</i> is at an interpolated position higher than or equal to <i>Position</i> . The position will consider active offset on <i>StartAxisId</i> .
12	CND_AXLABS	Reset when the axis <i>StartAxisId</i> is at an interpolated position lower than <i>Position</i> . The position must be construed as absolute.
13	CND_AXGABS	Reset when the axis <i>StartAxisId</i> is at an interpolated position higher than or equal to <i>Position</i> . The position must be construed as absolute.
14	CND_AXLph	Reset when the axis <i>StartAxisId</i> is at a real position lower than <i>Position</i> . The position will consider active offset on <i>StartAxisId</i> .
15	CND_AXGph	Reset when the axis <i>StartAxisId</i> is at a real position higher than or equal to <i>Position</i> . The position will consider active offset on <i>StartAxisId</i> .
16	CND_AXLABSpf	Reset when the axis <i>StartAxisId</i> is at a real position lower than <i>Position</i> . The position must be construed as absolute.
17	CND_AXGABSpj	Reset when the axis <i>StartAxisId</i> is at a real position higher than or equal to <i>Position</i> . The position must be construed as absolute.

By means of the “*Offset*” parameter we may define an offset to be associated with an axis during the reset operation. For example, if the “*Offset*” parameter is set to 50.0, when the position reset operation is activated, the axis will move to position 50.0 instead of 0.0. By means of the “*RollStep*” parameter we may define the modulus to be applied during the axis position reset. For instance, if a rotary axis is at 3610 degrees and we execute a GMC_SetResetQuote function with a “*RollStep*” of 360, upon the execution of the GMC_ActivateResetQuote function, the axis will move to a position of 10 degrees instead of 0.0.

The formula used to apply the GMC_ActivateResetQuote is as follows:



It is necessary to rely on a PLC interpolator to be able to set the axis reset position; however, the axis needs not necessarily belong to the interpolator specified, it may be one of OPENcontrol system axes associated with the PLC.

A position reset becomes active only when the position reset command, i.e., GMC ActivateResetQuote, is given.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0016002C	Axis ID not accepted
0x0016002B	Axis ID not found
0x00160031	Axis not related to the PLC
0x00160032	Wrong reset condition
0x0016005E	Wrong position axis rollover

Example:

See also:

GMC_ActivateResetQuote, GMC_SetResetQuote,
GMC_SetResetQuoteOnMarker, GMC_SetResetQuoteShift.

GMC_SetResetQuoteOnSignal,

3.79 GMC_SetResetQuoteONsignal

Function **GMC_SetResetQuoteONsignal** configures the axis reset point influenced by a signal.

Syntax:

```
ret_code := GMC_SetResetQuoteONsignal    (ExecMode, ExecStatus, InterId, Signal, AxisId,
                                         Mode, RollStep, Offset);
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	Interpolator identifier PLC [1..40]
<i>Signal (BOOL)</i>	Reset synchronising signal
<i>AxisId (INT)</i>	Axis identifier [1...64]
<i>Mode (INT)</i>	Reset mode
<i>RollStep (LREAL)</i>	Reset rollover step
<i>Offset (LREAL)</i>	Offset to be associated with reset

Execution mode:

Wait / NoWait

Use:

Through this function we configure, for the axis concerned, the parameters to be used when the “axis position reset” command is executed, to be executed, depends on a signal status upon the execution of function **GMC_ActivateResetQuote**. Through the “axis position reset” operation we execute the reset of the axis position, i.e., upon the execution of the **GMC_ActivateResetQuote** function, the axis moves to position 0. This operation is similar to the execution of an origin, but, unlike the latter, it is applied “on the go”; the system calculates its own internal origin, referred to as “reset origin”.

By means of the “*Mode*” and “*Signal*” parameters we may define when the reset operation is activated:

<i>Mode =</i>	CND_RAISE	Reset on rising edge of signal <i>Signal</i>
	CND_FAIL	Reset on falling edge of signal <i>Signal</i>
	CND_ON	Reset on “ON” level (true) of signal <i>Signal</i>
	CND_OFF	Reset on “OFF” level (false) of signal <i>Signal</i>

By means of the “*Offset*” parameter we may define an offset to be associated with an axis during the reset operation. For example, if the “*Offset*” parameter is set to 50.0, when the position reset operation is activated, the axis will move to position 50.0 instead of 0.0. By means of the “*RollStep*” parameter we may define the modulus to be applied during the axis position reset. For instance, if a rotary axis is at 3610 degrees and we execute a **GMC_SetResetQuote** function with a “*RollStep*” of 360, upon the execution of the **GMC_ActivateResetQuote** function, the axis will move to a position of 10 degrees instead of 0.0.

The formula activating the **GMC_ActivateResetQuote** is:

$$\begin{aligned} \text{New axis position} &= \text{Current position mod RollStep} + \text{Offset} && (\text{if RollStep is different from 0}) \\ \text{New axis position} &= \text{Offset} && (\text{if RollStep is 0}) \end{aligned}$$



A PLC interpolator has to be used to set the reset quote. Axis doesn't need to belong to the interpolator specified, but it may be one of the OPENcontrol axes currently associated with the PLC. Reset quote is inactive until the instruction 'GMC_ActivateResetQuote' is sent.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x00160031	Axis not associated with PLC
0x00160032	Reset condition wrong
0x00160036	Variable of synchronising signal wrong

Example:

```
Ret      : dword;
InterpId : INT;
...
(*Sets reset quote for axis ID 1 with value 100.0 to execute on ON level of the
signal M0_0 *)
Ret := GMC_SetResetQuoteONsignal (MODE_WAIT, ULO, InterpId, M0_0, 1,
                                  CND_ON, 0.0, 10.0) ;
```

See also:

[GMC_ActivateResetQuote](#), [GMC_SetResetQuoteOnPosition](#), [GMC_SetResetQuote](#),
[GMC_SetResetQuoteOnMarker](#), [GMC_SetResetQuoteShift](#).

3.80 GMC_SetResetQuoteShift

Function **GMC_SetResetQuoteShift** configures the value of reset axis position change.

Syntax:

```
ret_code := GMC_SetResetQuoteShift (ExecMode, ExecStatus, InterpId, AxisId, Shift, Offset) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>SpindleId</i> (INT)	Axis identifier [1...64]
<i>Shift</i> (LREAL)	Reset value change
<i>Offset</i> (LREAL)	Offset to associate with reset

Execution mode:

Wait / NoWait

Use:

Through this function we configure, for the axis concerned, the parameters needed to change the “reset origin”. This “reset origin” was determined during the execution of an earlier **GMC_ActivateResetQuote**, or is nil, if no **GMC_ActivateResetQuote** has been executed. The actual origin change will take place upon the execution of the next **GMC_ActivateResetQuote** function.

By means of the “*Offset*” parameter we may define an offset to be associated with an axis during the reset operation. For example, if the “*Offset*” parameter is set to 50.0, when the position reset operation is activated, the axis will move to position 50.0 instead of 0.0.



It is necessary to rely on a PLC interpolator to be able to set the axis reset position; however, the axis needs not necessarily belong to the interpolator specified, it may be one of **OPENcontrol** system axes associated with the PLC. A position reset becomes active only when the position reset command, i.e., **GMC_ActivateResetQuote**, is given.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0016002C	Axis ID not allowed
0x0016002B	Axis ID not found
0x00160031	Axes not related to the PLC

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Changes reset quote for axis ID 1 of a 50.0 value to execute on signal M0_0
level ON *)
Ret := GMC_SetResetQuoteONShift (MODE_WAIT, UL0, InterpId, 1, 50.0, 0.0);
```

See also: [GMC_ActivateResetQuote](#), [GMC_SetResetQuoteOnPosition](#),
[GMC_SetResetQuote](#),[GMC_SetResetQuoteOnMarker](#), [GMC_SetResetQuoteOnSignal](#).

3.81 GMC_Homing

Function GMC_Homing executes the axis reset.

Syntax:

```
ret_code := GMC_Homing (ExecMode, ExecStatus, InterpId, EndSignal, AxisId) ;
```

Input parameters.

<i>ExecMode</i> (INT)	Execution mode command [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Execution status command
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>AxisId</i> (INT)	Axis identifier [1...64]

Output parameter:

<i>EndSignal</i> (BOOL)	Boolean variable to give the end of activity signal (<i>EndSignal</i> set to TRUE)
-------------------------	---

Possible overload:

GMC_Homing (INT,WORD,INT,BOOL,INT[,INT]..)

Execution mode:

Wait / NoWait

Use:

This function makes it possible to execute the homing cycle for one or more axes (up to 12). At the end of the execution of the movement, the “*EndSignal*” is set to true. The execution mode of a homing cycle is described in detail in the application manual.

The reset stage makes no check on the operating limits set; they are activated by the system only at the end of the homing cycle. At the end of the homing cycle the reset value (see GMC_ActivateResetQuote) previously set for an axis, if any, is reset and only the TotalOffset remains active.

The movement is executed with the dynamic parameters of speed, acceleration/deceleration, jerk and minimum times specified for motion in Manual mode. All types of acceleration ramp are used.

When an axis is undergoing a reset stage, this is shown by bit _ZEROASS contained in the status word of the axis. The search direction of the reset micro is defined by bit _MICRONEG, also contained in the axis status word, where, at the end of the reset operation, we shall find bit _HC_DONE indicating that the reset has been executed successfully. To learn about the state of an axis, use function GMC_ReadAxisInfo.



Function CANNOT be programmed within a continuous mode.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not defined
0x00160003	Too many activities running
0x00160004	Move elements finished
0x00160005	Too many axes involved
0x0016002C	Unaccepted Axis ID
0x0016002B	Axis ID not found
0x0016002D	Axis ID referred to a gantry
0x00160027	Axis ID referred to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities finished
0x00160028	Axis already in use form another interpolator
0x00160029	Axis already in use from another interpolator in Hold
0x00160030	Axis already in use as slave from another interpolator
0x0016002A	Multiple homing multiple with axes having different clock
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160061	Wrong Homing direction
0x00160036	EndSignal synchronism variable wrong
0x0032005B	Option A48 disabled

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Executes multiple homing on axis ID 1 and ID 2 *)
Ret := GMC_Homing (MODE_WAIT, UL0, InterpId, M0_0, 1, 2) ;

```

See also:

[GMC_ReadAxisInfo](#), [GMC_ActivateResetQuote](#), [GMC_SetRamp](#), [GMC_SetJerk](#),

3.82 GMC_SpindleChangeGear

Function GMC_SpindleChangeGear performs the gear change on the spindle.

Syntax:

```
ret_code := GMC_SpindleChangeGear (ExecMode, ExecStatus, InterpId, SpindleId, Gear) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>SpindleId</i> (INT)	Axis identifier [1...64]
<i>Gear</i> (INT)	Gear to use [1..4]

Execution mode:

Wait / NoWait

Use:

Performs the gear change on the axis of a given spindle. The gear number must be between 1 and 4. The spindle may already be rotating in any mode.



In order to set a gear on the spindle it is necessary to rely on a PLC interpolator; in any event, the spindle needs not belong the interpolator specified.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x0016002C	Spindle ID not admitted
0x0016002B	Spindle ID not found
0x0016002D	Spindle ID with reference to a gantry
0x00160043	Spindle ID without reference to a gantry
0x00160045	Gear value wrong
0x00160062	Spindle orientation running

Example:

```
Ret      : dword;
InterpId : INT;
...
(*Sets gear 3 for spindle with ID 17 17 *)
Ret := GMC_SpindleChangeGear (MODE_WAIT, UL0, InterpId, 17, 3) ;
```

See also:

[Gmc_SpindleG97](#), [GMC_SpindleG96](#), [GMC_SpindleOrient](#)

3.83 GMC_SpindleG96

Function GMC_SpindleG96 activates the spindle rotation at a constant cutting speed.

Syntax:

```
ret_code := GMC_SpindleG96 (ExecMode, ExecStatus ,InterId, SpindleId, Speed, LowerSpeed,
                            UpperSpeed, AxisId) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	PLC interpolator identifier [1..40]
<i>SpindleId (INT)</i>	Axis identifier [1...64]
<i>Speed (LREAL)</i>	Constant cutting speed (mm/min)
<i>LowerSpeed (LREAL)</i>	Lower rotation speed accepted (revolutions/min)
<i>UpperSpeed (LREAL)</i>	Upper rotation speed accepted (revolutions/min)
<i>AxisId (INT)</i>	Reference axis identifier [1...64]

Execution mode:

Wait / NoWait

Use:

Starts spindle rotation at a specified speed defined as cutting speed on tool surface. The actual rotation speed of the spindle is calculated by the system based on the diameter of the piece (lathe) or the diameter of the tool (router/grinder). In order for the speed on the surface to be maintained constant with varying diameter size, speed in RPM must be determined as a function of the position of the *AxisId* axis which functions as an input parameter. The sign of the “*Speed*“ parameter defines the direction of spindle parameter; when the sign changes, the interpolator stops the spindle before restarting it in the opposite direction. The system takes into account the spindle override percentage requested. The spindle may be already rotating in any mode. If rotation speed exceeds the *LowerSpeed* or *UpperSpeed* limits, or if the maximum rotation speeds associated with the gear in use are exceeded, the fact that this limit has been exceeded is notified in the axis status (which can be read through GMC_ReadAxisInfo).

Return values:

<i>ret_code (HEX)</i>	<i>Description</i>
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x0016002C	Spindle or axis ID not admitted
0x0016002B	Spindle ID or axis not found
0x0016002D	Spindle or axis ID with reference to a gantry
0x00160043	Spindle or axis ID without reference to a gantry
0x00160027	Axis ID with reference to a spindle
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160045	Range value wrong
0x0016000C	Command aborted

0x0016000D	Interpolation activities have been used up
0x00160028	Spindle being used by another interpolator
0x00160029	Spindle being used by another interpolator in Hold
0x00160030	Spindle being used as Slave by another interpolator
0x0016004C	LowerSpeed value wrong
0x0016004D	UpperSpeed value wrong
0x0016004E	UpperSpeed and LowerSpeed parameters not congruent

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Sets spindle rotation with ID 17 referred to axis ID1, Cutting speed 3000.0
mm/min *)
Ret := GMC_SpindleG96 (MODE_WAIT, ULO, InterpId, 17, 3000.0,
                      100.0, 5000.0, 1) ;

```

See also:

[GMC_SpindleChangeGear](#), [GMC_SpindleG97](#), [GMC_SpindleOrient](#)

3.84 GMC_SpindleG97

Function GMC_SpindleG97 enables spindle rotation to a certain RPM speed.

Syntax

```
ret_code := GMC_SpindleG97 (ExecMode, ExecStatus ,InterpId, SpindleId, Speed) ;
```

Input parameters:

<i>ecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterpId (INT)</i>	PLC interpolator identifier [1..40]
<i>SpindleId (INT)</i>	Axis identifier [1...64]
<i>Speed (LREAL)</i>	Speed rotation (revolutions/min)

Execution mode:

Wait / NoWait

Use:

Starts spindle rotation at a specified speed defined in RPM (revolutions/min). The spindle may already be rotating in any mode. The sign of the “Speed” parameter defines the direction of spindle parameter; when the sign changes, the interpolator stops the spindle before restarting it in the opposite direction.

The system takes into account the spindle override percentage requested. If rotation speed exceeds the limits is notified in the axis status (which can be read through GMC_ReadAxisInfo).

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x0016002C	Unaccepted spindle ID
0x0016002B	Spindle ID not found
0x0016002D	Spindle ID referred to a gantry
0x00160043	Spindle ID not referred to a spindle
0x00160003	Too many activities running
0x00160004	Move elements finished
0x00160045	Range value wrong
0x0016000D	Interpolation activities finished
0x0016000C	Command aborted
0x00160028	Spindle in use by another interpolator
0x00160029	Spindle in use by another interpolator in Hold
0x00160030	Spindle in use as slave by another interpolator

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Sets spindle ID 17 to 3000.0 turns/min *)
Ret := GMC_SpindleG97 (MODE_WAIT, UL0, InterpId, 17, 3000.0) ;
```

See also:

[Gmc_SpindleChangeGear](#), [GMC_SpindleG96](#), [GMC_SpindleOrient](#)

3.85 GMC_SpindleOrient

Function GMC_SpindleOrient orients the spindle.

Syntax:

ret_code := GMC_SpindleOrient (ExecMode, ExecStatus ,InterId, SpindleId, Speed, Position)

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	PLC interpolator identifier [1..40]
<i>SpindleId (INT)</i>	Axis identifier [1...64]
<i>Speed (LREAL)</i>	Speed direction in mm/turn
<i>Position (LREAL)</i>	Spindle position (degrees)

Execution mode:

Wait / NoWait

Use:

Orientates the spindle (*SpindleId*) according to the angular position (degrees) defined by *Position* + the offset configured in ODM. The orientation stage is executed at *Speed* velocity. If *Speed* is 0, the velocity used is the one configured in ODM related to the current range. Using GMC_ReadAxisInfo on the axis status, it is possible to determine if the spindle has been already oriented or is in orientation stage. Any other function active on the spindle remove the information.

Return values:

<i>ret_code (HEX)</i>	Description
0x00000000	Function executed without errors
0x00160014	Interpolator missing
0x0016002C	Unaccepted axis or spindle ID
0x0016002B	Axis or spindle ID not found
0x0016002D	Axis or spindle ID referred to a gantry
0x00160043	Spindle ID not referred to a spindle
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016000D	Interpolation activities finished
0x0016000C	Command aborted
0x00160028	Spindle in use by another interpolator
0x00160029	Spindle in use by another interpolator in Hold
0x00160030	Spindle in use as slave by another interpolator

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Orientates the spindle to a 90 degrees position with Vel 100 turns per
minute*)
Ret := GMC_SpindleOrient (MODE_WAIT, UL0, InterpId, 17, 100.0, 90.0) ;
```

See also: [GMC_SpindleChangeGear](#), [GMC_SpindleG97](#), [GMC_SpindleG96](#)

3.86 GMC_WaitONposition

Function GMC_WaitONposition waits on axis position.

Syntax:

```
ret_code := GMC_WaitONposition (InterpId, AxisId, Position, Mode) ;
```

Input parameters:

<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>AxisId</i> (INT)	Axis identifier [1...64] on which to synchronise
<i>Position</i> (LREAL)	Synchronism position
<i>Mode</i> (INT)	Synchronism mode

Execution mode:

Wait

Use:

By means of this function waits until axis “*AxisId*” respects the condition “*Mode*” in comparison to position “*Position*”:

Mode = 10 CND_AXL	Starts when axis AxisId is at an interpolated position lower than Position. The position must take into account the offsets active on the AxisId axis.
11 CND_AXG	Starts when axis AxisId is at an interpolated position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis.
12 CND_AXLABS	Starts when axis AxisId is at an interpolated position lower than Position. This position must be construed as absolute.
13 CND_AXGABS	Starts when axis AxisId is at an interpolated position higher than or equal to Position. This position must be construed as absolute.
14 CND_AXLph	Starts when axis AxisId is at a real position lower than Position. The position must take into account the offsets active on the AxisId axis.
15 CND_AXGph	Starts when axis tAxisId is at a real position higher than or equal to Position. The position must take into account the offsets active on the AxisId axis.
16 CND_AXLABSp	Starts when axis AxisId is at a real position lower than Position. This position must be construed as absolute.
17 CND_AXGABSpj	Starts when axis AxisId is at a real position higher than or equal to Position. This position must be construed as absolute.



Function CANNOT be programmed within a continuous stage.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Interpolator not present
0x00160003	Too many activities in progress
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016000D	Interpolation activities have been used up
0x00160032	Synchronism position wrong
0x0016005E	Rollover axis position wrong

Example:

```

Ret      : dword;
InterpId : INT;
...
(* Waits until axis ID 5 is higher than 200.0 *)
Ret := GMC_WaitONposition (InterpId, 5, 200.0, CND_AXGABS) ;

```

See also:

[GMC_WaitONvariable](#), [GMC_WaitONSIGNAL](#), [GMC_WaitONTIMEOUT](#)

3.87 GMC_WaitONsignal

Function GMC_WaitONsignal waits for value of a signal.

Syntax:

```
ret_code := GMC_WaitONsignal (InterpId, Signal, Mode) ;
```

Input parameters:

<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>Signal</i> (INT)	Boolean variable of synchronism
<i>Mode</i> (INT)	Synchronism mode

Execution mode:

Wait

Use:

Through this function we wait for the “*Signal*” signal to assume a value as defined by the “*Mode*” parameter. It can be:

Mode =	1 CND_RAISE	Waiting for rising edge of Signal
	2 CND_FAIL	Waiting for falling edge of Signal
	3 CND_ON	Waiting for ON level (true) of Signal
	4 CND_OFF	Waiting for OFF level (false) of Signal



Function CANNOT be programmed within a continuous.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016000D	Interpolation activities finished
0x00160032	Wrong synchronism condition
0x00160036	Wrong Signal variable

Example:

```
Ret      : dword;
InterpId : INT;
...
(*Waits for falling edge of M0_0 *)
Ret := GMC_WaitONsignal (InterpId, M0_0, CND_FAIL) ;
```

See also:

GMC_WaitONposition, GMC_WaitONvariable, GMC_WaitONtimeout

3.88 GMC_WaitONtimeout

Function GMC_WaitONtimeout waits for a certain time.

Syntax:

```
ret_code := GMC_WaitONtimeout (InterpId, Timeout) ;
```

Input parameters:

InterpId (INT)	Interpolator identifier PLC [1..40]
Timeout (DWORD)	Waiting time

Execution mode:

Wait

Use:

This function defines a certain waiting time, expressed in a unit of 10 ms.



Function CANNOT be programmed within a continuous stage.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016000D	Interpolation activities finished

Example

```
Ret      : dword;
InterpId : INT;
...
(* Waits for 20 ms*)
Ret := GMC_WaitONsignal (InterpId, 2) ;
```

See also:

[GMC_WaitONposition](#), [GMC_WaitONvariable](#), [GMC_WaitONsignal](#)

3.89 GMC_WaitONvariable

Function GMC_WaitONvariable waits for a value of a variable.

Syntax:

```
ret_code := GMC_WaitONvariable (Interpld, Variable, VarValue, Mode) ;
```

Input parameters:

<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>Variable</i> (INT)	Synchronism variable
<i>VarValue</i> (INT)	Synchronism values
<i>Mode</i> (INT)	Synchronism mode

Execution mode:

Wait

Use:

By means of this function you are waiting for the variable “*Variable*” to take a value as defined by parameter “*Mode*”:

StopMode =	40 CND_VRLESS	Waits for Variable to become lower than VarValue
	41 CND_VRGREAT	Waits for Variable to become higher than VarValue
	42 CND_VREQUAL	Waits for Variable to become equal to VarValue
	43 CND_VRDIFF	Waits for Variable to become different from VarValue



Function CANNOT be programmed within a continuous stage.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016000D	Interpolation activities finished
0x00160032	Wrong synchronism condition
0x00160048	Wrong synchronism variable

Example:

```
Ret      : dword;
Interpld : INT;
...
(* Waits until MI300 is 200.0 *)
Ret := GMC_WaitONvariable (Interpld, MI300, 200.0, CND_VREQUAL) ;
```

See also:

GMC_WaitONposition, GMC_WaitONSignal, GMC_WaitONtimeout

3.90 GMC_Drill

Function GMC_Drill executes the drilling cycle.

Syntax:

```
ret_code := GMC_Drill(ExecMode, ExecStatus, Interpld, EndSignal, AxisId, Position, IniPos,  
                  RetPos, RetMode, DrilFeed, Dwell) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	PLC interpolator identifier [1..40]
<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Position</i> (LREAL)	Axis position
<i>IniPos</i> (LREAL)	Start cycle position
<i>RetPos</i> (LREAL)	End cycle position
<i>RetMode</i> (INT)	Return mode
<i>DrilFeed</i> (LREAL)	Drilling speed
<i>Dwell</i> (LREAL)	Dwell time (in seconds)
<i>Feed</i> (FeedDescr_Struct)	Feed rate

Output parameters:

<i>EndSignal</i> (BOOL)	End drilling signal
-------------------------	---------------------

Execution mode:

Wait / NoWait

Use:

This function allows the drilling cycle on an axis. When drilling cycle stops, “*EndSignal*” is set on true. “*AxisId*” and “*Position*” CAN’T be duplicated in order to request the drilling on one axis only. “*Position*” parameter can have several meanings according to the axis programming context. If programming is absolute, it refers to the final position of the axis at the end of the motion, but if incremental, *Position* refers to the space axis has to run starting from the current position. All positions refer to the origin activated on the programmed axis. If there are any working limits active, they will be checked before the motion execution generating (in some circumstances)the Over travel error.



Function cannot be programmed within a continuous stage.

Drilling cycle

Cycle dynamics executions (accelerations/decelerations, jerk and minimum ramp times)depends on the Feed input structure. This structure defines velocity, acceleration, deceleration and jerk values used in the motion. If this filed are left blank, motion will be executed according to a default dynamic.

The following table describes *Feed* structure fields:

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed Jerk (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated only for linear axes

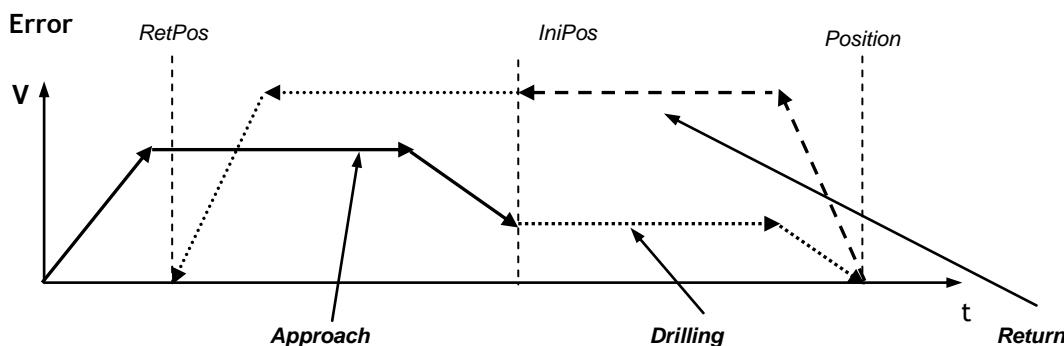
Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Acceleration minimum ramp time (ms)
TminDec	LREAL	Deceleration minimum ramp time (ms)

Below the values *TypeFeed* can have:

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Drilling cycle can be divided in stages as follow:



- **Approach:** it's the shift from the current point to the approach point *IniPos*. Motion is executed at rapid feed and with dynamic working parameters unless different programming in the Feed structure.
- **Drilling:** It's a motion at drill feed *DrillFeed* up to "Position". When this position is reached, motion dwells for a while (*Dwell* parameter).

- **Return (normal mode):** this phase is executed with only one motion to reach the point “*RetPos*” at rapid feed and with dynamic work parameters, unless different Feed structure programming.
- **Return (slow mode):** this phase is in two motions. The first move set the axis on a point defined by *IniPos* (start drilling point) at *DrillFeed*, the second motion places it on a point defined by *RetPos* with rapid feed and dynamic work parameters, unless different Feed structure programming.

<i>RetMode</i>	<i>Mnemonic</i>	<i>Description</i>
0	CYC_RETNORM	Executes one return phase in rapid
1	CYC_RETSLOW	Splits the return in two stages: work+rapid.

Return values:

<i>ret_code (HEX)</i>	<i>Description</i>
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016002C	Unaccepted ID axis
0x0016002B	ID axis not found
0x0016002D	ID axis referred to a gantry
0x00160027	ID axis referred to a spindle
0x0016000D	Interpolation activities finished
0x00160028	Axis already in use from another interpolator
0x00160029	Axis already in use from another interpolator in Hold
0x00160030	Axis used as slave by another interpolator
0x00160040	Axis on positive over travel
0x00160041	Axis on negative over travel
0x0016003C	Axis exceeds Software positive limit
0x0016003D	Axis exceeds Software negative limit
0x00160046	Null speed
0x00160036	Wrong EndSignal or InvSignal synchronism variable
0x0016005F	Cycle execution directions are not consistent
0x00160060	Cycle execution positions are not consistent
0x0032005B	Option A48 disabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(* Drills axis ID 1 at 100.0 position with V 500.0, approaches at 50.0 and
returns at 0.0 with only one motion, 50 milliseconds dwelling on the hole *)
Ret := GMC_Drill (MODE_WAIT, UL0, InterpId, M0_0, 1, 100.0,
                  50.0, 0.0, CYC_RETNORM, 500.0, 0.05, Feed) ;

```

3.91 GMC_Bore

Function GMC_Bore executes the boring cycle.

Syntax:

<i>ret_code</i> := GMC_Bore	<i>(ExecMode, ExecStatus, InterId, EndSignal, StopSignal, AxisId, Position, IniPos, RetPos, RetMode, SpinId, BoreFeed, Dwell, Feed)</i> ;
-----------------------------	---

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	Interpolator identifier PLC [1..40]
<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Position</i> (LREAL)	Axis position
<i>IniPos</i> (LREAL)	Start cycle position
<i>RetPos</i> (LREAL)	End cycle position
<i>RetMode</i> (INT)	Return mode
<i>SpinId</i> (INT)	Spindle identifier [1..64]
<i>BoreFeed</i> (LREAL)	Boring feed
<i>Dwell</i> (LREAL)	Dwell time (in seconds)
<i>Feed</i> (FeedDescr_Struct)	Feed rate

Output parameters:

<i>EndSignal</i> (BOOL)	End boring signal
<i>StopSignal</i> (BOOL)	Stop spindle signal

Execution mode:

Wait / NoWait

Use:

This function allows the execution of a boring cycle on axis. When the cycle ends, “*EndSignal*” is set true. Parameters “*AxisId*” and “*Position*” CAN’T be duplicated to request the boring cycle on one axis only.

“*Position*” can have different meanings according to the axis programming. In absolute programming, it refers to the final position where the axis is at the end of the motion whereas, in incremental programming, defines the space axis has to cover starting from the current position.

All positions refers to the origin on the axis programmed. If software limits are enabled, they will be checked before the motion, possibly generating an Over travel error.



Function cannot be programmed within a continuous stage.

Boring cycle

Cycle dynamics executions (accelerations/decelerations, jerk and minimum ramp times)depends on the Feed input structure. This structure defines velocity, acceleration, deceleration and jerk values used in the motion. If this filed are left blank, motion will be executed according to a default dynamic.

The following table describes *Feed* structure fields:

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed values (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed jerk (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated only on linear axes

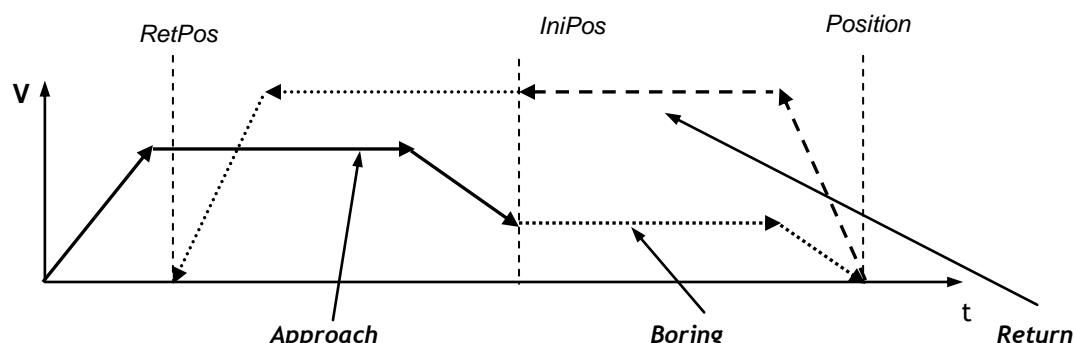
Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration Jerk (unit/s ³)
TminAcc	LREAL	Minimum ramp time acceleration (ms)
TminDec	LREAL	Minimum ramp time deceleration (ms)

Below all the values *TypeFeed* variable can have:

TypeFeed = 0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Boring cycle stages can be resumed as follow:



- **Approach:** it's a shift from the current point towards *IniPos* (approach point). Motion is executed at rapid feed and with working dynamic parameters, unless different Feed structure programming.
- **Boring:** it's a motion at boring feed (*BoreFeed*) up to "Position". When the position is reached, motion dwells for a while (*Dwell*). At the same time, on *StopSignal* (at true) is set the request to stop the spindle and PLC is forced to stop the spindle rotation. When spindle is stopped ad dwell time is over, the return phase is enabled.
- **Return (normal mode):** this phase is executed in one motion from the point identified *RetPos* at rapid feed and dynamic working parameters, unless different programming in the Feed structure.
- **Return (slow mode):** this phase is divided in two motions. The first moves towards the point defined by *IniPos* (boring starting point) at *BoreFeed* feed rate, whereas the second moves towards the point defined by *RetPos* at rapid feed and with working dynamic parameters (unless

different Feed structure programming).

RetMode	Mnemonic	Description
0	CYC_RETNORM	Executes one return phase in rapid
1	CYC_RETSLOW	Splits the return in two stages: work+rapid.

- **End:** when the cycle ends, the spindle is requested to restart resetting the *StopSignal* (*at false*) value. PLC will restart the spindle rotation, but system won't wait its "homing".

Return values:

The following table resumes the possible values *ret_code* can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016002C	Unaccepted ID axis
0x0016002B	ID axis not found
0x0016002D	ID axis referred to a gantry
0x00160027	ID axis referred to a spindle
0x0016000D	Interpolation activities finished
0x00160028	Axis already in use from another interpolator
0x00160029	Axis already in use from another interpolator in Hold
0x00160030	Axis used as slave by another interpolator
0x00160040	Axis on positive over travel
0x00160041	Axis on negative over travel
0x0016003C	Axis exceeds Software positive limit
0x0016003D	Axis exceeds Software negative limit
0x00160046	Null speed
0x00160036	Wrong EndSignal or InvSignal synchronism variable
0x0016005F	Cycle execution directions not consistent
0x00160060	Cycle execution positions not consistent
0x0016002A	Axis and spindle have a different clock
0x00160043	Spindle ID not related to any spindle
0x0032005B	Option A48 disabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
... (* Bores axis ID 1 at 100.0 position with V 300.0, approaches
      at 50.0 and returns at 0.0 with only one movement, without waiting.
      On M0_1 manages stop spindle signal with Id 10 *)
Ret := GMC_Bore (MODE_WAIT, ULO, InterpId, M0_0, M0_1, 1, 100.0,
                  50.0, 0.0, CYC_RETNORM, 10, 300.0, 0.0, Feed) ;

```

3.92 GMC_Tapp

Function GMC_Tapp enables a tapping cycle without transducer.

Syntax:

```
ret_code := GMC_Tapp (ExecMode, ExecStatus, InterId, EndSignal, InvSignal,
AxisId, Position, IniPos, RetPos, SpinId, TapFeed, TRPValue, Feed) ;
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	Interpolator identifier PLC [1..40]
<i>AxisId (INT)</i>	Axis identifier [1..64]
<i>Position (LREAL)</i>	Axis position
<i>IniPos (LREAL)</i>	Start cycle position
<i>RetPos (LREAL)</i>	End cycle position
<i>SpinId (INT)</i>	Spindle identifier [1..64]
<i>TapFeed (LREAL)</i>	Tapping feed rate
<i>TRPValue (LREAL)</i>	Return feed percentage in 0,01 %
<i>Feed (FeedDescr_Struct)</i>	Feed rate

Output parameters:

<i>EndSignal (BOOL)</i>	End tapping cycle
<i>InvSignal (BOOL)</i>	Spindle inversion signal

Execution mode:

Wait / NoWait

Use:

This function allows the execution of a tapping cycle without transducer (on the spindle involved in the cycle)on a single axis. When the cycle ends, “EndSignal” is set true. Parameters “AxisId” and “Position” CAN’T be duplicated to request the boring cycle on one axis only.

“Position” can have different meanings according to the axis programming. In absolute programming, it refers to the final position where the axis is at the end of the motion whereas, in incremental programming, defines the space axis has to cover starting from the current position.

All positions refers to the origin on the axis programmed. If software limits are enabled, they will be checked before the motion, possibly generating an Over travel error.



Function can be programmed within a continuous stage.

Tapping cycle execution

Cycle dynamics executions (accelerations/decelerations, jerk and minimum ramp times)depends on the Feed input structure. This structure defines velocity, acceleration, deceleration and jerk values used in the motion. If this filed are left blank, motion will be executed according to a default dynamic.

The following table describes the *Feed* structure fields.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed jerk (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated for linear axes only

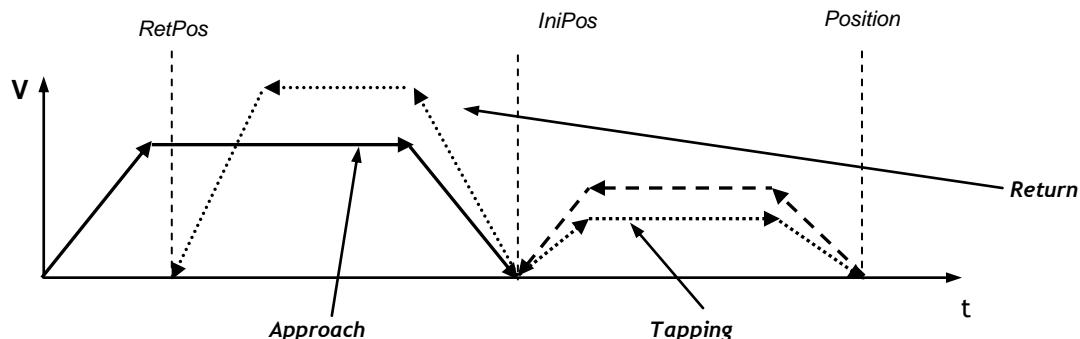
Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration Jerk (unit/s ³)
TminAcc	LREAL	Minimum ramp time acceleration (ms)
TminDec	LREAL	Minimum ramp time deceleration (ms)

Below the values *TypeFeed* variable can have:

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Tapping cycle stages can be resumed as follow:



- ▷ **Approach:** it's the shift from the current point to the approach point *IniPos*. Motion is executed at rapid feed and with dynamic working parameters unless different programming in the Feed structure.
- ▷ **Tapping:** it's a motion at tapping feed (*TapFeed*) up to “*Position*”. Before reaching this position, on the *InvSignal* (*at true*) variable is set the spindle rotation inversion. PLC is forced to invert the spindle rotation direction. When axis reaches the final position “*Position*”, the return phase is enabled.
- ▷ **Tapping (return):** it's a motion at tapping feed *TRPValue* multiplied by *TapFeed* factor up to the *IniPos* position. Before finishing the return phase, on the *InvSignal* (*a false*) variable is signalled the request to restore the original spindle rotation direction. PLC is forced to restore the spindle rotation direction.

- **Return (normal mode):** this phase is executed with only one motion to reach the point “*RetPos*” at rapid feed and with dynamic work parameters, unless different Feed structure programming.

Tapping feed rate is calculated by the formula:

$$\text{TapFeed} = \text{S} * \text{p} * 0.9$$

where **S** is the spindle rotation speed, **p** is the tap pitch and 0.9 is the feed decreasing factor to maintain the tension of the spring of the compensated tap tool holder.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016002C	Unaccepted ID axis
0x0016002B	ID axis not found
0x0016002D	ID axis referred to a gantry
0x00160027	ID axis referred to a spindle
0x0016000D	Interpolation activities finished
0x00160028	Axis already in use from another interpolator
0x00160029	Axis already in use from another interpolator in Hold
0x00160030	Axis used as slave by another interpolator
0x00160040	Axis on positive over travel
0x00160041	Axis on negative over travel
0x0016003C	Axis exceeds Software positive limit
0x0016003D	Axis exceeds Software negative limit
0x00160046	Null speed
0x00160036	Wrong EndSignal or InvSignal synchronism variable
0x0016005F	Cycle execution directions not consistent
0x00160060	Cycle execution positions not consistent
0x0016002A	Axis and spindle have a different clock
0x00160043	Spindle ID not related to any spindle
0x0032005B	Option A48 disabled

Example:

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(* Taps axis ID 1 at 100.0 position with V 20.0, approaches
   at 50.0 and returns at 0.0 with an 120% feed rate.
   On M0_1 manages the spindle inversion signal with Id 10 *)
Ret := GMC_Tapp (MODE_WAIT, ULO, InterpId, M0_0, M0_1, 1, 100.0,
                  50.0, 0.0, 10, 20.0, 12000, Feed) ;

```

See also:

GMC_TappTransd, GMC_Thread

3.93 GMC_TappTransd

Function GMC_TappTransd enables the tapping cycle with transducer.

Syntax:

```
ret_code := GMC_TappTransd (ExecMode, ExecStatus ,Interpld, EndSignal, InvSignal, AxisId,  

Position, IniPos, RetPos, SpinId, Pitch, TKGValue,  

TAGValue, Feed) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>AxisId</i> (INT)	Axis ID [1..64]
<i>Position</i> (LREAL)	Axis position
<i>IniPos</i> (LREAL)	Starting cycle position
<i>RetPos</i> (LREAL)	End cycle position
<i>SpinId</i> (INT)	Spindle ID [1..64]
<i>Pitch</i> (LREAL)	Tapping pitch
<i>TKGValue</i> (LREAL)	Hard tapping gain at constant speed
<i>TAGValue</i> (LREAL)	Hard tapping gain while accelerating
<i>Feed</i> (FeedDescr_Struct)	Feed rate

Output parameters:

<i>EndSignal</i> (BOOL)	End tapping signal
<i>InvSignal</i> (BOOL)	Spindle inversion signal

Execution mode:

Wait / NoWait

Use:

This function allows the execution of a tapping cycle with transducer (on the spindle involved in the cycle) on a single axis. When the cycle ends, “*EndSignal*” is set true. Parameters “*AxisId*” and “*Position*” CAN’T be duplicated to request the boring cycle on one axis only.

“*Position*” can have different meanings according to the axis programming. In absolute programming, it refers to the final position where the axis is at the end of the motion whereas, in incremental programming, defines the space axis has to cover starting from the current position. All positions refers to the origin on the axis programmed. If software limits are enabled, they will be checked before the motion, possibly generating an Over travel error.



Function can be programmed within a continuous stage.

Tapping cycle with transducer execution

Cycle dynamics executions (accelerations/decelerations, jerk and minimum ramp times)depends on the Feed input structure. This structure defines velocity, acceleration, deceleration and jerk values used in the motion. If this filed are left blank, motion will be executed according to a default dynamic.

The following table describes *Feed* structure fields:

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed jerk (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated only for linear axes

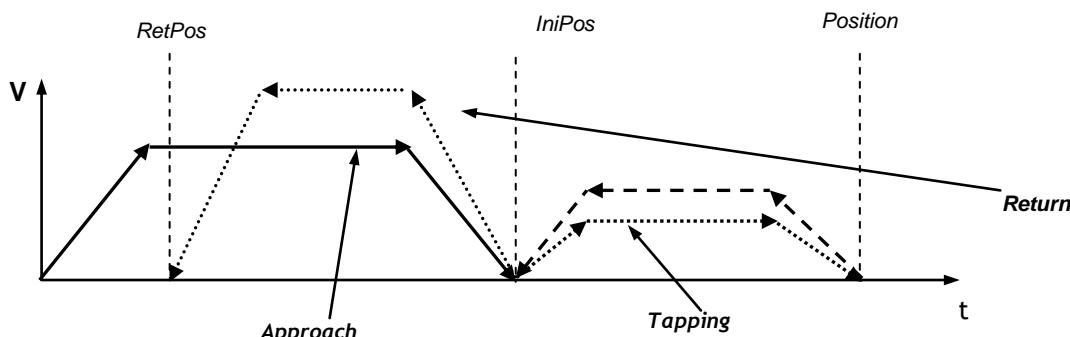
Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration Jerk (unit/s ³)
TminAcc	LREAL	Minimum ramp time acceleration (ms)
TminDec	LREAL	Minimum ramp time deceleration (ms)

Below the values *TypeFeed* variable can have:

TypeFeed	Type	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Tapping cycle can be resumed as follow:



- ▷ **Approach:** it's the shift from the current point to the approach point *IniPos*. Motion is executed at rapid feed and with dynamic working parameters unless different programming in the Feed structure.
- ▷ **Tapping:** this phase finishes when the position “*Position*” is reached and has a tapping feed motion calculating according to the spindle rotation feed and the taping pitch (*Pitch*) programmed. Before reaching this position, on variable *InvSignal* (a *true*) is signalled the request of spindle rotation motion inversion. PLC has to invert the spindle rotation direction. When axis reaches the final position (*Position*) the return phase is activated.
- ▷ **Tapping (return):** tapping speed is calculated according to the spindle rotation feed and the tapping pitch (*Pitch*) programmed to run backwards the previous tapping. Before finishing the return phase, the request of restoring the original spindle direction is signalled on *InvSignal* (at *false*) variable. After this instruction PLC restores the spindle rotation direction.
- ▷ **Return (normal mode):** this phase is executed with only one motion to reach the point “*RetPos*” at rapid feed and with dynamic work parameters, unless different Feed structure programming.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016002C	Unaccepted ID axis
0x0016002B	ID axis not found
0x0016002D	ID axis referred to a gantry
0x00160027	ID axis referred to a spindle
0x0016000D	Interpolation activities finished
0x00160028	Axis already in use forform another interpolator
0x00160029	Axis already in use forform another interpolator in Hold
0x00160030	Axis used as slave by another interpolator
0x00160040	Axis on positive over travel
0x00160041	Axis on negative over travel
0x0016003C	Axis exceeds Software positive limit
0x0016003D	Axis exceeds Software negative limit
0x00160046	Null speed
0x00160036	Wrong EndSignal or InvSignal synchronism variable
0x0016005F	Cycle execution directions not consistent
0x00160060	Cycle execution positions not consistent
0x0016002A	Axis and spindle have a different clock
0x00160043	Spindle ID not related to any spindle
0x0032005B	Option A48 disabled

Example:

```
Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(* Taps axis ID 1 at 100.0 position, approaches
   t 50.0 and returns at 0.0 with 25.0 pitch.
   On M0_1 manages spindle inversion signal with Id 10 *)
Ret := GMC_TappTransd (MODE_WAIT, UL0, InterpId, M0_0, M0_1, 1, 100.0,
                      50.0, 0.0, 25.0, 1.5, 1.0, Feed) ;
```

See also:

[GMC_Tapp](#), [GMC_Thread](#)

3.94 GMC_Thread

Function GMC_Thread executes threading cycle.

Syntax:

```
ret_code := GMC_Thread (ExecMode, ExecStatus ,InterId, EndSignal, SpinId, Pitch,
PitchVariation, Deviation, TKGValue, TAGValue, AxisId, Position,Feed);
```

Input parameters

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_NOWAIT_ACK]
<i>ExecStatus (DWORD)</i>	Command execution status
<i>InterId (INT)</i>	Interpolator identifier PLC [1..40]
<i>SpinId (INT)</i>	Spindle identifier [1..64]
<i>Pitch (LREAL)</i>	Threading pitch
<i>PitchVariation (LREAL)</i>	Pitch variation
<i>Deviation (LREAL)</i>	Starting deviation (degrees)
<i>TKGValue (LREAL)</i>	Hard tapping gain at constant feed
<i>TAGValue (LREAL)</i>	Hard tapping gain during acceleration
<i>AxisId (INT)</i>	Axes [1..64] identifier for up to 12 axes
<i>Position (LREAL)</i>	Final position
<i>Feed (FeedDescr_Struct)</i>	Feed rate

Output parameter

<i>EndSignal (BOOL)</i>	Tapping ended
-------------------------	---------------

Possible overload:

GMC_ThreadTransd (INT, DWORD, INT, BOOL, INT, LREAL, LREAL, LREAL, LREAL,
FeedDescr_Struct , INT, LREAL[, INT, LREAL]...)

Execution mode:

Wait / NoWait

Use:

This function allows a threading motion execution on several axes. When execution ends, “EndSignal” is set to true. “AxisId” and “Position” can be duplicated to request a synchronous motion on up to 12 axes. Feed motion is calculated according to the spindle feed rotation, to the threading pitch (Pitch) programmed and with all the dynamic parameters related to accelerations/decelerations, jerks and minimum ramp time. this structure defines feed, acceleration, deceleration and jerk values, used within the motion. If these fields are not filled, motion will be executed with default dynamic.

The following table defines *Feed* fields structure:

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min) (Not used in threading)
Accel	LREAL	Programmed acceleration (unit/s ²)
Decel	LREAL	Programmed deceleration (unit/s ²)
Jerk	LREAL	Programmed jerk (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated for linear axes only

Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration Jerk (unit/s ³)
TminAcc	LREAL	Minimum ramp time acceleration (ms)
TminDec	LREAL	Minimum ramp time deceleration (ms)

Below the values *TypeFeed* can have:

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

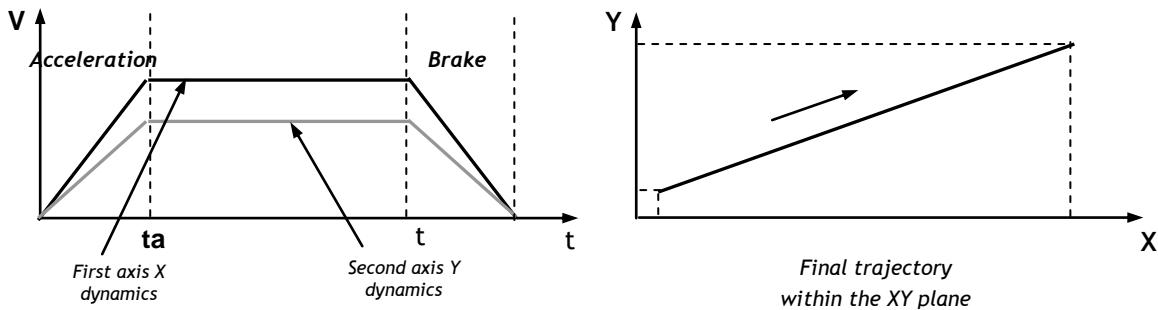
“Position” parameter can have several meanings according to the axis programming context. If programming is absolute, it refers to the final position of the axis at the end of the motion, but if incremental, *Position* refers to the space axis has to run starting from the current position. All positions refer to the origin activated on the programmed axis. If there are any working limits active, they will be checked before the motion execution generating (in some circumstances) the Over travel error.



Function can be programmed within a continuous mode.

Threading

Axes defined in GMC_Thread start and finish motion at the same time, therefore are “interpolated”, maintaining the trajectory required by the motion. Each axis will have its own feed diagram, which is not affected by feeds, accelerations and jerks of other axes. As lot of axes are interpolated, the acceleration phase start at the same time for all.



For each axis a check is made to avoid dynamics higher than the ones configured for the axis (feed, acceleration and jerk). If for the axis are required dynamics too difficult, dynamics have to be reduced in the axes involved in the motion. All axes involved in the same motion have the same servo clock; when there are axes with different clocks, motions have to be executed separately.

For threading with decreasing pitch, the calculation has to be made for the starting pitch, the pitch variation and the threading length, in order to have the pitch higher than zero before reaching the final position.

Return values:

The following table resumes all values *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Missing interpolator
0x00160003	Too many activities running
0x00160004	Move elements finished
0x0016002C	Unaccepted ID axis
0x0016002B	ID axis not found
0x0016002D	ID axis referred to a gantry
0x00160027	ID axis referred to a spindle
0x0016000D	Interpolation activities finished
0x00160028	Axis already in use from another interpolator
0x00160029	Axis already in use from another interpolator in Hold
0x00160030	Axis used as slave by another interpolator
0x00160040	Axis on positive over travel
0x00160041	Axis on negative over travel
0x0016003C	Axis exceeds Software positive limit
0x0016003D	Axis exceeds Software negative limit
0x00160046	Null speed
0x00160036	Wrong EndSignal or InvSignal synchronism variable
0x0016005F	Cycle execution directions not consistent
0x00160060	Cycle execution positions not consistent
0x0016002A	Axis and spindle have a different clock
0x00160043	Spindle ID not related to any spindle
0x0032005B	Option A48 disabled

Example:

```
Ret      : dword;
InterpId : INT;
...
(* Threading executed at constant pitch with axes ID 1 and 2 at 100.0 position,
100,0 with pitch 25.0. Spindle has Id 10. Threading starts with a
45 degrees Deviation *)
Ret := GMC_Thread (MODE_WAIT, UL0, InterpId, M0_0, 10, 25.0, 0,0, 45.0,
1.5, 1.0, 45.0, 1, 100.0, 2, 100.0) ;
```

3.95 GMC_Circle3DCCW

GMC_Circle3DCCW function executes an anticlockwise circular motion in space given the centre and the final point.

Syntax

```
ret_code := GMC_Circle3DCCW (ExecMode, ExecStatus.InterpId, EndSignal, Feed, NX, NY, NZ,  
                  CenterX, CenterY, CenterZ, AxisId, Position) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	Interpolator identifier PLC [1..40]
<i>Feed</i> (FeedDescr_Struct)	Feed motion
<i>NX</i>	Abscissa component of the vector perpendicular to the plane containing the arc
<i>NY</i>	Ordinate component of the vector perpendicular to the plane containing the arc
<i>NZ</i>	Vertical component of the vector perpendicular to the plane containing the arc
<i>CenterX</i>	Abscissa axis centre coordinate
<i>CenterY</i>	Ordinate axis centre coordinate
<i>CenterZ</i>	Vertical axis centre coordinate
<i>AxisId</i> (INT)	Axis identifier [1...64] for up to 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters

<i>EndSignal</i> (BOOL)	Boolean variable where setting (at TRUE) the activity end
-------------------------	---

Possible overloads

GMC_Circle3DCCW (INT, DWORD, INT, BOOL, FeedDescr_Struct, LREAL, LREAL, LREAL, LREAL, LREAL, LREAL, INT, LREAL, INT, LREAL[,INT,LREAL]...)

Execution mode

Wait / NoWait

Use

This function allows a counter clockwise motion within space (only with three axes programmed) or a helicoidal (if more than three axes are programmed) having a known centre. The first three axes to be programmed are abscissa, ordinate and vertical axis. When execution ends, “EndSignal” is set to true. “AxisId” and “Position” can be duplicated to request a synchronous motion on up to 12 axes (3 minimum). Motion can be executed whether at rapid feed or at the feed defined by “Feed” parameter. This structure defines feed, acceleration, deceleration and jerk values, used within the motion. If these fields are not filled, motion will be executed with default dynamic.

The following table describes Feed structure fields.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value(unit/min)
Accel	LREAL	Acceleration (unit/s ²)
Decel	LREAL	Deceleration (unit/s ²)
Jerk	LREAL	Jerk in acceleration (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated for linear axes only

Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Jerk in acceleration (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

The following table lists all possible values *TypeFeed* variable can have.

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Speed is to be conceived as vector speed, i.e., the rate of displacement of the set of axes moved, not that of the individual axes. Execution speed is limited based on the size of the radius to be executed, so as to limit the centrifugal acceleration that is generated by the circular motion. The limit is imposed by considering the acceleration of either the X or the Y axis, whichever is smaller.

$$\text{Feed} < \sqrt{A * \text{Radius}} \quad A = \min(\text{accX}, \text{accY}, \text{accZ})$$

“Position” parameter can have several meanings according to the axis programming context. If programming is absolute, it refers to the final position of the axis at the end of the motion, but if incremental, *Position* refers to the space axis has to run starting from the current position. All positions refer to the origin activated on the programmed axis. If there are any working limits active, they will be checked before the motion execution generating (in some circumstances) the Over travel error.

Origins and absolute/incremental programming mode are considered also for “CenterX”, “CenterY” and “CenterZ”; “CenterX”, refers to the first axis programmed, “CenterY”, refers to the second axis programmed and “CenterZ”, refers to the third axis programmed.

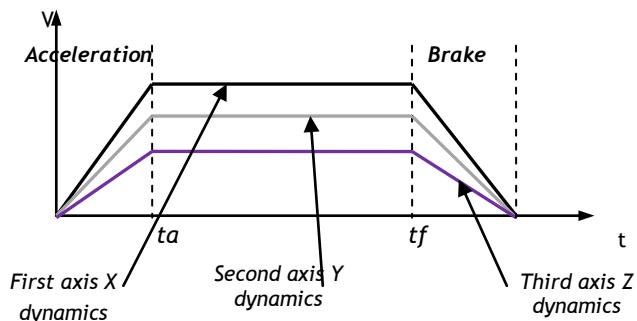
As “NX”, “NY” and “NZ” parameters are vector components, the absolute programming must be always considered. These parameters allow to identify the circumference plane in case of a 180° or 360° programmed arc. In case of a vector not properly defined (that is there is no plane perpendicular to the vector containing the programmed circumference) the control will calculate the plane of setting the “N” non-orthogonal. “NX”, “NY” and “NZ” values are not considered for all arcs different from 180° and 360°.



Function can be programmed within a continuous mode.

Coordinating the axes of a movement

The axes defined in GMC_Circle3DCCW *start and end their movements simultaneously*; hence they are mutually “interpolated” and therefore the required trajectory is maintained. From the dynamic standpoint each axis will adopt a speed curve of its own, independent of the other axes in terms of feed rates reached and accelerations and jerks adopted; since the motions of the various axes are mutually interpolated, the acceleration stage ends simultaneously for all the axes; similarly, the deceleration stage begins simultaneously.



In any event, checks are carried out to determine whether the speed curves requested for each axis exceed the (feed, acceleration and jerk) values configured for the axis in question. If the speed curves requested for an axis are too high, limits are set on the speed curves of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes with different clocks will have to be moved separately.

Programmed feed

This function works according to the principle of “feed on the profile”, i.e., the speed obtained for the movement as a whole; by this principle, the speed at which an axis moves is calculated based on speed on the profile and the motion component of the individual axis relative to the overall distance to be covered “on the profile”

Circumferences

The programming of circular motions (also referred to as circumferences) is susceptible to programming inaccuracies (numerical representation) and hence the circles programmed are often rejected in as much as they do NOT describe a perfect circle. To remedy this problem, “tolerances” and “methodologies” are provided for use in defining the circles. In this connection, see function GMC_ToleranceParams.

Full circles

In programming a full circle, the end point must be made to coincide with the starting point. There is a system variable, **FullCirc (FCT)**, by which if the distance between the starting point and the end point is smaller than this constant, the circle executed will be a full circumference. For example, assuming our circle has a starting point (100, 100, 100) and an end point (100.1,100.1,100.1), it will be executed as an arc if **FullCirc=0.1**, and will be executed as a full circle (with end point 100,100, 100) if **FullCirc=0.15**, the distance between the two points being 0.14142.

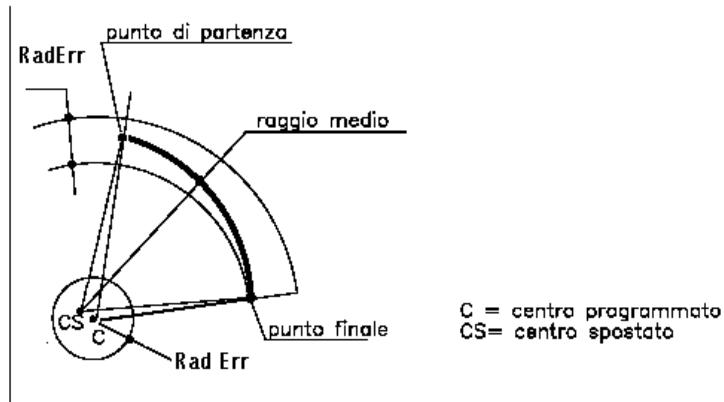
Circles execution mode

Another problem to be addressed in connection with the programming of circular motions is the precision required in programming the centre in order to describe a perfect circumference. First of all, in programming a circumference, we determine the radii subtended between the starting point and the end point: if the differences between the radii is nil, the circle is executed as requested, if the difference exceeds a given threshold, as defined by system variable **RadErr (CET)**, an error is generated, if the difference is in the 0 - RadErr range, methodologies are provided for recalculating the circumference parameters so that the circumference may be defined in a congruent manner.

The **RadMode (ARM)** system variable 4 different programming modes that allow to:

- ▷ Recalculate the centre of a circumference within a tolerance defined by **RadErr**
- ▷ Move the starting point within a tolerance defined by **RadErr**
- ▷ Recalculate the centre of a circle irrespective of **RadErr** (default mode)
- ▷ Maintain the centre and move the starting/ end points within a tolerance defined by **RadErr**

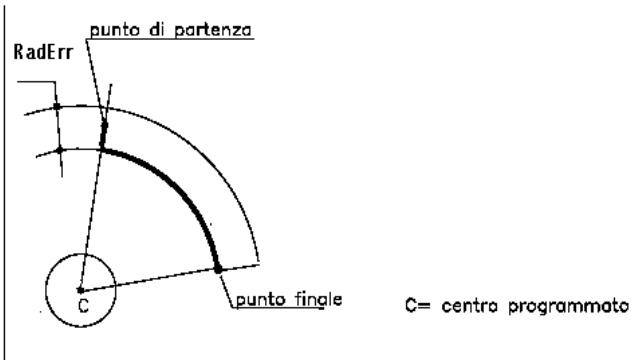
RadMode =0 Recalculation of the centre of a circumference within a certain tolerance



A circumference passing through the starting and end points is executed having its centre within a tolerance defined by parameter **RadErr**.

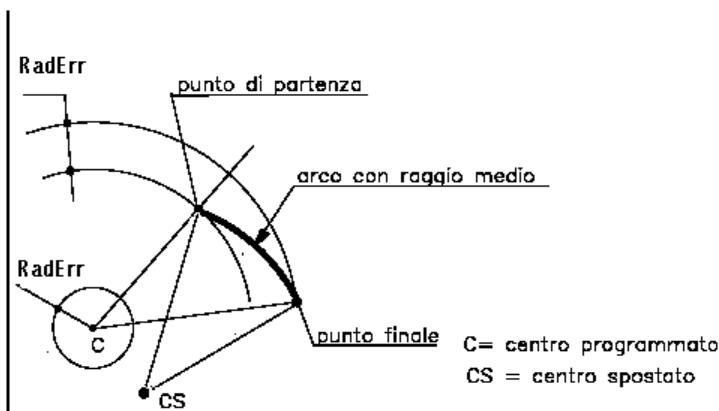
In this case the circumference is executed with a mean radius.

RadMode =1 Moving the starting point to within a certain tolerance



The circumference executed passes through a starting point corrected to fall within the tolerance defined with **RadErr** and an end point as programmed. The centre remains unchanged.

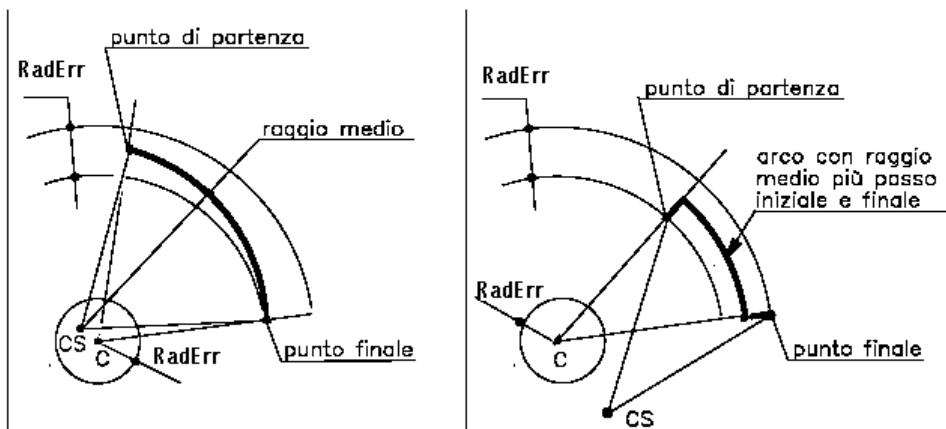
RadMode =2 Recalculation of the centre of the circumference



The circumference passes through the starting point and the end point and its centre is moved without considering the tolerance. In this case the circumference is executed with a mean radius.

RadMode =3 The centre is maintained and the starting and end points are moved

C = centro programmato
 CS = centro spostato



If the movement of the centre remains within the tolerance range defined by RadErr , then the circumference executed passes through the starting and end points and has its centre moved to a new position (same as with $\text{RadMode}=0$). If the movement is not within the tolerance specified by RadErr , then the centre remains where it was and the starting and end points are corrected by a value corresponding to $\text{RadErr}/2$.

Return values

The following table lists all possible values *ret_code* variable can have:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160003	Too many activities running
0x00160014	Missing interpolator
0x00160004	Movement elements have been used up
0x00160005	Too many axes involved
0x0016002C	Unaccepted axis id
0x0016002B	Axis id not found
0x0016002D	Axis id referred to a gantry
0x00160027	Axis id referred to a spindle
0x0016002F	Axis id duplicated
0x0016000D	Interpolation activities finished

0x00160028	Axis already used forform another interpolator
0x00160029	Axis already used forform another interpolator in Hold
0x00160030	Axis already used as a Slave forform another interpolator
0x0016002A	Axes with different clock axes
0x0016003C	Axis exceeding the positive Software limit
0x0016003D	Axis exceeding the negative Software limit
0x00160040	Axis on positive over travel
0x00160041	Axis on negative over travel
0x00160046	Null feed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x001A0011	Vector orthogonal to the plane containing the circle is not defined
0x0032005B	Option A48 disabled

Example

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
NX       : LREAL ;
NY       : LREAL ;
NZ       : LREAL ;
...
(* Executes a CCW circular motion at 10000 mm/min feed in position 100 for
first, second and third axis, centre is at 50.5, 50.0, 50.0;on the plane
perpendicular to the vector is designed a 180° arc
(SIN(45), -COS(45),0).
When motion finishes, M0_0 variable is set on true *)
```



```

Feed.Value    := 10000.0 ;
Feed.TypeFeed := MODE_WORK;
NX = SIN(45 * PIGRECA/180.0) ;
NY = - COS(45 * PIGRECA/180.0) ;
NZ = 0.0 ;
Ret := GMC_Circle3DCCW (MODE_WAIT, ULO, InterpId, M0_0, Feed, NX, NY, NZ, 50.0,
50.0, 50.0, 1, 100.0, 2, 100.0, 3, 100.0) ;
```

See also

[GMC_SetRamp](#), [GMC_SetJerk](#), [GMC_SetFeedOverride](#), [GMC_AbsoluteQuote](#), [GMC_IncrementalQuote](#), [GMC_ActivateOrigin](#), [GMC_ToleranceParms](#)

3.96 GMC_Circle3D3P

GMC_Circle3D3P function executes a circular motion for three given points.

Syntax

```
ret_code := GMC_Circle3D3P (ExecMode, ExecStatus, Interpld, EndSignal, Feed,
                           P3X, P3Y, P3Z, AxisId, Position);
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>Interpld</i> (INT)	Interpolator identifier PLC [1..40]
<i>Feed</i> (FeedDescr_Struct)	Feed rate
<i>P3X</i>	Midpoint coordinate abscissa axis
<i>P3Y</i>	Midpoint coordinate ordinate axis
<i>P3Z</i>	Midpoint coordinate vertical axis
<i>AxisId</i> (INT)	Axis identifier [1...64] for up to 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters

<i>EndSignal</i> (BOOL)	Boolean variable where signalling (at TRUE) the activity end
-------------------------	--

Possible overloads

GMC_Circle3D3P (INT, DWORD, INT, BOOL, FeedDescr_Struct, LREAL, LREAL, LREAL, INT, LREAL, INT, LREAL[,INT,LREAL]...)

Execution mode

Wait / NoWait

Use

This function allows to execute a movement, which is either circular (if only three axes have been programmed) or helicoidal (more than 3 axes programmed), given three points not aligned in space. The three axes that are programmed first are taken to be the abscissa, the ordinate and the vertical axes. At the end of the execution of the movement, signal “*EndSignal*” is set to true. Parameters “*AxisId*” and “*Position*” may be duplicated in order to request the synchronous movement of several axes, up to a maximum of 12 (3 as a minimum).

The movement is executed at rapid feed or at the feed defined by the “*Feed*” parameter. This structure defines speed, acceleration, deceleration and jerk values which will be used during movement. If these fields are not set, the movement will be executed by default dynamics.

The following table describes Feed structure fields.

Field	Type	Description
TypeFeed	BYTE	Feed type
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Acceleration (unit/s ²)
Decel	LREAL	Deceleration (unit/s ²)
Jerk	LREAL	Jerk in acceleration (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated for linear axes only

Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Jerk in acceleration (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

The following table lists all possible values *TypeFeed* variable can have

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Speed is to be conceived as vector speed, i.e., the rate of displacement of the set of axes moved, not that of the individual axes. Execution speed is limited based on the size of the radius to be executed, so as to limit the centrifugal acceleration that is generated by the circular motion. The limit is imposed by considering the acceleration of either the X or the Y axis, whichever is smaller.

$$\text{Feed} < \sqrt{A * \text{Radius}} \quad A = \min(\text{accX}, \text{accY}, \text{accZ})$$

“Position” parameter can have several meanings according to the axis programming context. If programming is absolute, it refers to the final position of the axis at the end of the motion, but if incremental, *Position* refers to the space axis has to run starting from the current position. All positions refer to the origin activated on the programmed axis. If there are any working limits active, they will be checked before the motion execution generating (in some circumstances) the Over travel error.

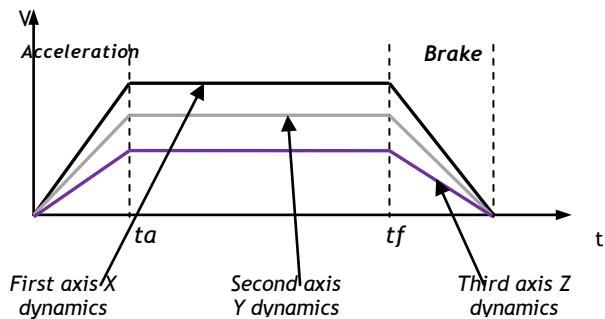
Absolute/incremental programming modes and origins are considered also for “P3X”, “P3Y” and “P3Z” parameters. The system calculates the plane where there is the arc and this is the plane passing by three given points. With this technique a complete circle cannot be programmed as the circumference plane wouldn’t be univocally determined.



Function can be programmed within a continuous cycle.

Coordinating the axes of a movement

The axes defined in GMC_Circle3D3P start and end their movements simultaneously; hence they are mutually “interpolated” and therefore the required trajectory is maintained. From the dynamic standpoint each axis will adopt a speed curve of its own, independent of the other axes in terms of feed rates reached and accelerations and jerks adopted; since the motions of the various axes are mutually interpolated, the acceleration stage ends simultaneously for all the axes; similarly, the deceleration stage begins simultaneously.



In any event, checks are carried out to determine whether the speed curves requested for each axis exceed the (feed, acceleration and jerk) values configured for the axis in question. If the speed curves requested for an axis are too high, limits are set on the speed curves of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes with different clocks will have to be moved separately.

Programmed feed

The function works according to the principle of “feed on the profile”, i.e., the speed obtained for the movement as a whole; by this principle, the speed at which an axis moves is calculated based on speed on the profile and the motion component of the individual axis relative to the overall distance to be covered “on the profile”

Return values

The following table lists all values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160003	Too many activities in progress
0x00160014	Interpolator not present
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities have been used up
0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x001A0007	Three points aligned
0x0032005B	Option A48 disabled

Example

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
...
(* Executes a circular motion at 10000 mm/min feed rate at point 100 for the
first axis, 100 for the second axis and 100 for the third axis, and passing by
50.0, 70.0, 80.0 points;
At end of movement variable M0_0 is set to true *)

Feed.Value    := 10000.0 ;
Feed.TypeFeed := MODE_WORK;
Ret := GMC_Circle3D3P (MODE_WAIT, ULO, InterpId, M0_0, Feed, 50.0, 70.0, 80.0,
1, 100.0, 2, 100.0, 3, 100.0) ;

```

See also

[GMC_SetRamp](#), [GMC_SetJerk](#), [GMC_SetFeedOverride](#), [GMC_AbsoluteQuote](#), [GMC_IncrementalQuote](#), [GMC_ActivateOrigin](#), [GMC_ToleranceParms](#)

3.97 GMC_Circle3DCW

GMC_Circle3DCW function executes a clockwise circular motion in space known the centre and the final point.

Syntax

```
ret_code := GMC_Circle3DCW (ExecMode, ExecStatus, InterpId, EndSignal, Feed, NX, NY, NZ,  
                          CenterX, CenterY, CenterZ, AxisId, Position) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>Feed</i> (FeedDescr_Struct)	Feed motion
<i>NX</i>	Abscissa component of the vector perpendicular to the plane containing the arc
<i>NY</i>	Ordinate component of the vector perpendicular to the plane containing the arc
<i>NZ</i>	Vertical component of the vector perpendicular to the plane containing the arc
<i>CenterX</i>	Abscissa axis centre coordinate
<i>CenterY</i>	Ordinate axis centre coordinate
<i>CenterZ</i>	Vertical axis centre coordinate
<i>AxisId</i> (INT)	Axis identifier [1...64] for up to 12 axes
<i>Position</i> (LREAL)	Final position

Output parameters

<i>EndSignal</i> (BOOL)	Boolean variable where signal (at TRUE) the activities end
-------------------------	--

Possible overloads

GMC_Circle3DCW (INT, DWORD, INT, BOOL, FeedDescr_Struct, LREAL, LREAL, LREAL, LREAL, LREAL, LREAL, INT, LREAL, INT, LREAL[,INT,LREAL]...)

Execution mode

Wait / NoWait

Use

This function allows to execute a counter clockwise movement, which is either circular (if only three axes have been programmed) or helicoidal (more than 32 axes programmed), whose centre is located at a given point. The first three axes that are programmed are taken to be the abscissa, the ordinate and the vertical axes. At the end of the execution of the movement, signal “*EndSignal*” is set to true. Parameters “*AxisId*” and “*Position*” may be duplicated in order to request the synchronous movement of several axes, up to a maximum of 12 (3 as a minimum).

The movement is executed at the feed defined by the “*Feed*” parameter or at rapid feed. This structure defines speed, acceleration, deceleration and jerk values which will be used during movement. If these fields are not set, the movement will be executed by default dynamics.

The following table describes Feed structure fields.

Field	Type	Description
TypeFeed	BYTE	TypeFeed
Value	LREAL	Feed value (unit/min)
Accel	LREAL	Acceleration (unit/s ²)
Decel	LREAL	Deceleration (unit/s ²)
Jerk	LREAL	Jerk in acceleration (unit/s ³)
VelOnlyLin	BOOL	Velocity calculated for linear axes only

Starting from the OPENcontrol 2.1 release, following information will be also available:

Field	Type	Description
DeJerk	LREAL	Deceleration jerk (unit/s ³)
TminAcc	LREAL	Minimum acceleration ramp time (ms)
TminDec	LREAL	Minimum deceleration ramp time (ms)

The following table lists all possible values *TypeFeed* variable can have:

TypeFeed	Mnemonic	Description
0	MODE_RAPID	Movements will use the rapid speed rate and the dynamic parameters of the rapid or programmed movements.
1	MODE_WORK	Movements will use the programmed speed and the dynamic parameters of programmed or work movement
2	MODE_FEED_t	Movements are programmed in time and will use the dynamic parameters of work or programmed movements.
3	MODE_FEED_1T	Movements are programmed in reverse time and will use the dynamic parameters of the work or programmed movements.

Speed is to be conceived as vector speed, i.e., the rate of displacement of the set of axes moved, not that of the individual axes. Execution speed is limited based on the size of the radius to be executed, so as to limit the centrifugal acceleration that is generated by the circular motion. The limit is imposed by considering the acceleration of either the X or the Y axis, whichever is smaller.

$$\text{Feed} < \sqrt{A * \text{Radius}} \quad A = \min(\text{accX}, \text{accY}, \text{accZ})$$

“Position” parameter can have several meanings according to the axis programming context. If programming is absolute, it refers to the final position of the axis at the end of the motion, but if incremental, *Position* refers to the space axis has to run starting from the current position. All positions refer to the origin activated on the programmed axis. If there are any working limits active, they will be checked before the motion execution generating (in some circumstances) the Over travel error.

The origins and the absolute/incremental programming modes are considered also for “CenterX”, “CenterY” and “CenterZ” parameters; “CenterX” refers to the first axis programmed, “CenterY” refers to the second axis programmed and “CenterZ” refers to the third axis programmed.

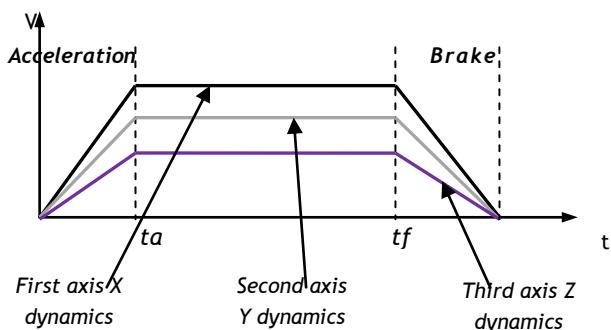
As “NX”, “NY” and “NZ” parameters are vector components, the absolute programming must be always considered. These parameters allow to identify the circumference plane in case of a programmed arc large as 180° or 360° . In case of vector not properly defined (no plane perpendicular to the vector and containing the programmed circumference can exists) the CN will calculates the plane offsetting the non-orthogonal “N”. For all arcs different from 180° and from 360° the “NX”, “NY” and “NZ” values are not considered.



Function can be programmed within a continuous cycle.

Coordinating the axes of a movement

The axes defined in GMC_Circle3DCW *start and end their movements simultaneously*; hence they are mutually “interpolated” and therefore the required trajectory is maintained. From the dynamic standpoint each axis will adopt a speed curve of its own, independent of the other axes in terms of feed rates reached and accelerations and jerks adopted; since the motions of the various axes are mutually interpolated, the acceleration stage ends simultaneously for all the axes; similarly, the deceleration stage begins simultaneously.



In any event, checks are carried out to determine whether the speed curves requested for each axis exceed the (feed, acceleration and jerk) values configured for the axis in question. If the speed curves requested for an axis are too high, limits are set on the speed curves of the other axes involved in the movement. All the axes involved in a movement must have the same servo clock; axes with different clocks will have to be moved separately.

Programmed feed

This function works according to the principle of “speed on the profile”, i.e., the speed obtained for the movement as a whole; by this principle, the speed at which an axis moves is calculated based on speed on the profile and the motion component of the individual axis relative to the overall distance to be covered “on the profile”

Circumferences

The programming of circular motions (also referred to as circumferences) is susceptible to programming inaccuracies (numerical representation) and hence the circles programmed are often rejected in as much as they do NOT describe a perfect circle. To remedy this problem, “tolerances” and “methodologies” are provided for use in defining the circles. In this connection, see function GMC_ToleranceParams.

Full circles

In programming a full circle, the end point must be made to coincide with the starting point. There is a system variable, **FullCirc (FCT)**, by which if the distance between the starting point and the end point is

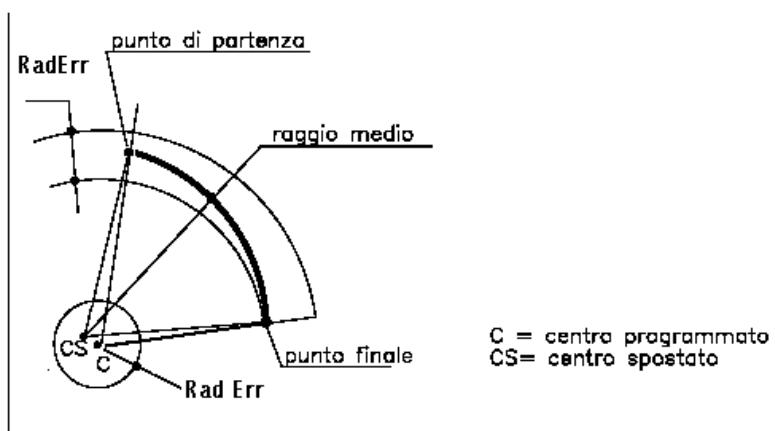
smaller than this constant, the circle executed will be a full circumference. For example, assuming our circle has a starting point (100, 100, 100) and an end point (100.1, 100.1, 100.1), it will be executed as an arc if FullCirc=0.1, and will be executed as a full circle (with end point 100,100, 100) if FullCirc=0.15, the distance between the two points being 0.14142.

Circle execution modalities

Another problem to be addressed in connection with the programming of circular motions is the precision required in programming the centre in order to describe a perfect circumference. First of all, in programming a circumference, we determine the radii subtended between the starting point and the end point: if the differences between the radii is nil, the circle is executed as requested, if the difference exceeds a given threshold, as defined by system variable **RadErr (CET)**, an error is generated, if the difference is in the 0 - RadErr range, methodologies are provided for recalculating the circumference parameters so that the circumference may be defined in a congruent manner. Four different programming modes, defined by means of system variable **RadMode (ARM)**, are available, to:

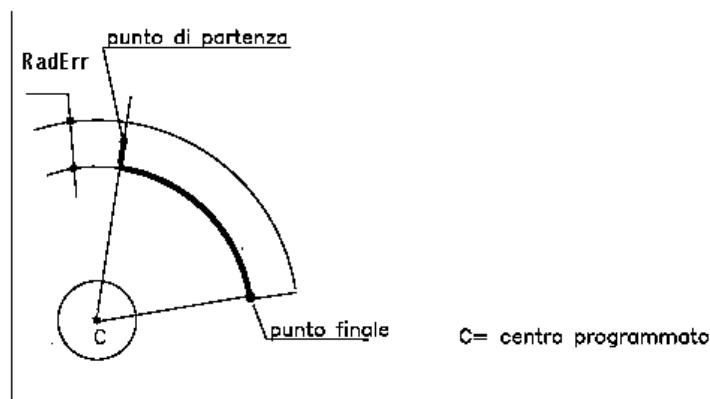
- ▷ Recalculate the centre of a circle within the tolerance defined by RadErr
- ▷ Move the starting point to within the tolerance defined by RadErr
- ▷ Recalculate the centre of a circle irrespective of RadErr (default mode)
- ▷ Maintain the centre where it is and move the starting and end points to within the tolerance defined by RadErr

RadMode =0 Recalculating the centre of a circle within a certain tolerance



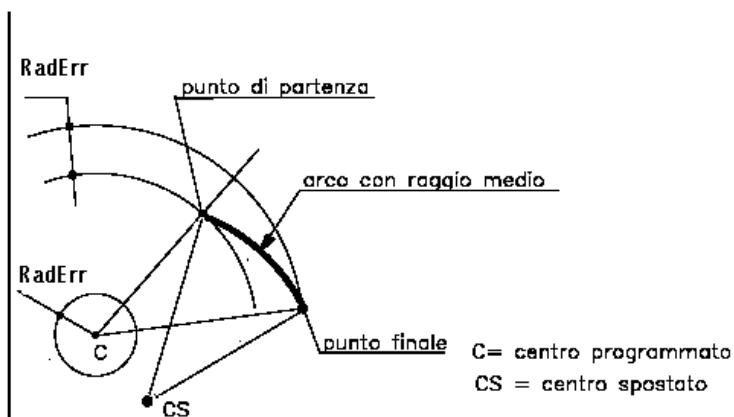
The circumference is recalculated so that it passes through the starting and end points and its centre is moved to within the tolerance range defined by parameter RadErr. In this case the circle is executed with a mean radius.

RadMode =1 Starting point moved to within a certain tolerance



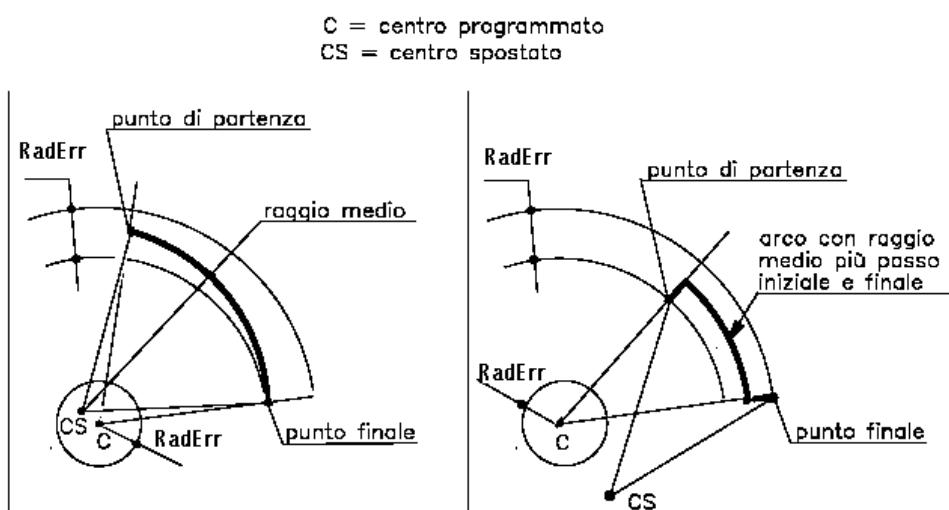
The circumference is recalculated so that it passes through a starting point moved to within the tolerance range defined by parameter RadErr and through the end point programmed. The centre is not moved.

RadMode =2 Recalculating the centre of a circle



The circumference is recalculated so that it passes through the starting and end points and its centre is moved regardless of tolerance constraints. In this case the circle is executed with a mean radius.

RadMode =3 Keeping the centre where it is and moving the starting and end points



If the movement of the centre of the circle takes place within the tolerance range defined by RadErr , then the circumference is recalculated so that it passes through the starting and end points and its centre is moved appropriately. (same as in RadMode=0). If

the movement of the circle is not within the tolerance range specified by RadErr , then the centre is maintained where it is and the starting and end points are corrected by a value corresponding to $\text{RadErr}/2$

Return values

The following table lists all possible values *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160003	Too many activities in progress
0x00160014	Interpolator not present
0x00160004	Movement entities have been used up
0x00160005	Too many axes involved
0x0016002C	Axis ID not admitted
0x0016002B	Axis ID not found
0x0016002D	Axis ID related to a gantry
0x00160027	Axis ID related to a spindle
0x0016002F	Axis ID duplicated
0x0016000D	Interpolation activities have been used up

0x00160028	Axis being used by another interpolator
0x00160029	Axis being used by another interpolator in Hold
0x00160030	Axis being used as Slave by another interpolator
0x0016002A	Axes with axes of different clock
0x0016003C	Axis beyond positive software limit
0x0016003D	Axis beyond negative software limit
0x00160040	Axis on positive limit switch
0x00160041	Axis on negative limit switch
0x00160046	Null speed
0x00160036	EndSignal synchronism variable wrong
0x00160005	Too many axes specified
0x001A0011	Vector orthogonal to the plane containing the circle is not defined
0x0032005B	Option A48 disabled

Example

```

Ret      : dword;
InterpId : INT;
Feed     : FeedDescr_Struct;
NX       : LREAL ;
NY       : LREAL ;
NZ       : LREAL ;
...
(*Executes a clockwise circular movement at a feed rate of 10000
mm/min at point 100 for the first axis and point, at 100 for the second and
100for the third, the centre is at 50.0, 50.0, 50.0, a 180° arc is executed on
the plane perpendicular to the vector(SIN(45), -COS(45),0).
At end of movement variable M0_0 is set to true*)

Feed.Value    := 10000.0 ;
Feed.TypeFeed := MODE_WORK;
NX = SIN(45 * PIGRECA/180.0) ;
NY = - COS(45 * PIGRECA/180.0) ;
NZ = 0.0 ;
Ret := GMC_Circle3DCW (MODE_WAIT, ULO, InterpId, M0_0, Feed, NX, NY, NZ, 50.0,
50.0, 50.0, 1, 100.0, 2, 100.0, 3, 100.0) ;

```

See also

GMC_SetRamp, GMC_SetJerk, GMC_SetFeedOverride, GMC_AbsoluteQuote, GMC_IncrementalQuote, GMC_ActivateOrigin, GMC_ToleranceParms

3.98 GMC_SetDynamicOverride

GMC_SetDynamicOverride sets Dynamic Override percentage and mode for all axes specified.

Syntax

```
ret_code := GMC_SetDynamicOverride (ExecMode, ExecStatus, InterpId, Mode,  
                          AxisId, OverValue) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>Mode</i> (BYTE)	Override activation mode
<i>AxisId</i> (INT)	Axis identifier [1...64]
<i>OverValue</i> (LREAL)	Override reference value

Possible overloads

GMC_SetDynamicOverride(INT,DWORD,INT,BYTE,INT,LREAL,[INT,LREAL],...)

Execution mode

Wait / NoWait

Use

This function is used to check in real time the axes feedrate dynamics. The dynamics is controlled according to the Mode parameter.

Mode = DYN_CLEAR (0) disables the Dynamic Override function on the axis; same as programming a 100% override.

In **Mode = DYN_PERC** (1), there is a % reduction of the dynamics (feed rate), the jerk will be simultaneously reduced of a percentage of the passed value (granularity 1/100 of % that is 10000 = 100%). This operation is executed only in case of a motion by the specified axis.

In **Mode = DYN_VALUE** (2) the dynamic is reduced to have the speed rate value equal to the value considered as parameter. In this case, an internal % reduction is calculated to perform the override required. In this case too, acceleration and jerk are simultaneously reduced of a percentage according to the re-calculated value.

As override is considered ONLY as a dynamic reduction factor instead of an incremental factor, % values higher than 100% will be limited to 100% while feed values higher than the present feed will be reduced to the present feed.

When an axis is interpolated with other axes, dynamic changes made by the GMC_SetDynamicOverride function affects also other axes not depending from the Dynamic Override.



In order to set the override % is necessary to use a PLC interpolator, in any case the axes may not belong to defined interpolator; here can be also called all the OPENcontrol axes, provided that they belong to the PLC

Return values

The following table lists all possible values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Undefined interpolator
0x00160031	Axis not associated to the PLC
0x0016002C	Unaccepted axis id
0x0016002B	Axis id not found
0x00160005	Too many axes specified

Example

```
Ret      : dword;
InterpId : INT;
...
(* Sets % at 50 for axes having ID 1 and 2 *)
Ret := GMC_SetDynamicOverride (MODE_WAIT, UL0, InterpId, DYN_PERC,
                               1, 50, 2, 50) ;
```

3.99 GMC_SetDynamicOverrideALL

GMC_SetDynamicOverrideALL function selects the Dynamic Override percentage and mode for all axes associated to the PLC.

Syntax

```
ret_code := GMC_SetDynamicOverrideALL (ExecMode, ExecStatus, InterId, Mode, OverValue) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ExecStatus</i> (DWORD)	Command execution status
<i>InterId</i> (INT)	PLC interpolator identifier [1..40]
<i>Mode</i> (BYTE)	Override activation mode
<i>OverValue</i> (LREAL)	Override reference values

Execution mode

Wait / NoWait

Use

This function is used for a real-time monitoring of the axes feed rate referred to a PLC or, when Mode field is in OR with the DYN_SINGLE_INTERP value, the dynamic of the interpolator specified. Dynamics is controlled according to the Mode parameter.

In **Mode = DYN_CLEAR** (0) Override Dynamic function on the axis is disabled; same as programming a 100% override.

In **Mode = DYN_PERC** (1), there is a % reduction of the dynamics (feed rate), the jerk will be simultaneously reduced of a percentage of the passed value (granularity 1/100 of % that is 10000 = 100%). This operation is executed only in case of a motion by the specified axis.

In **Mode = DYN_VALUE** (2) the dynamic is reduced to have the speed rate value equal to the value considered as parameter. In this case, an internal % reduction is calculated to perform the override required. In this case too, acceleration and jerk are simultaneously reduced of a percentage according to the re-calculated value.

As override is considered ONLY as a dynamic reduction factor instead of an incremental factor, % values higher than 100% will be limited to 100% while feed values higher than the present feed will be reduced to the present feed.



In order to set the override % is necessary to use a PLC interpolator, in any case the axes may not belong to defined interpolator; here can be also called all the OPENcontrol axes, provided that they belong to the PLC

Return values

The following table lists all possible values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160014	Undefined interpolator

Example

```
Ret      : dword;
InterpId : INT;
...
(* Sets % at 50 for all axes *)
Ret := GMC_SetDynamicOverrideALL (MODE_WAIT, ULO, InterpId, DYN_PERC,
                                  50.0);
```

3.100 GMC_CamAxiDefine

The function GMC_CamAxiDefine defines an electronic cam associated to a master and to a slave axis.

Syntax

```
ret_code := GMC_CamAxiDefine (InterpId, MasterId, MstType,, SlaveId, FileName, out CamId) ;
```

Input parameters

<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>MasterId</i> (INT)	Master identifier
<i>MstType</i> (INT)	Master type
<i>SlaveId</i> (INT)	Slave axis identifier [1..64]
<i>FileName</i> (STRING)	File string containin the correspondence between a slave and its master

Output parameter

<i>CamId</i> (INT)	Electronic cam identifier [1..64]
--------------------	-----------------------------------

Execution mode

Wait

Uso

The function GMC_CamAxiDefine creates an association between a slave and a master. This association is preset in tables in the form of positions that the slave must get according to the master positions. It is possible to create several electronic cams associated to the same master and to the same slave, simply by changing the input file.

MasterId is an identifier of the master that can be an interpolated axis (physical or virtual), a transducer or a MD variable. The field *MstType* can therefore have the values below:

<i>MstType</i>	<i>Mnemonic</i>	Description
0	CAM_MSTINT	Interpolated axis
1	CAM_MSTTRAS	Transducer
2	CAM_MSTVAR	MD variable

Tables are created tarting from a file in .csv format. The file is organized by strings, therefore each string refers to a cam position. The first string contains the infromation related to the file type and tp the number of strings compiled. There can be two types of file_:

A file having two coulmns where the first contains the master positions and the second ythe corresponding positions of its slave.

First string →	0;4	FileType; Strings number
Other strings →	100;10 150;13 200;18 250;10	Master;Slave

A file with several columns where the first contains the master positions and the others the polynomial coefficients from which to get the corresponding positions of the slave.

First string →	1;4	FileType; Strings number
Other strings →	100;1;0.2;3.3;4;5 150;;;;; 200;;;;; 250;;;;;	Master;pSlv0;pSlv1;pSlv2;pSlv3;pSlv4;pSlv5

When the cam is enabled, the master position within the table is searched and it is calculated the position of the slave axis lineary interpolating (or using polynomials) the two points including the master position. The positions of the master axis must be strictly monotone increasing or decreasing.



It is necessary to use an interpolator PLC in order to execute commands for managing cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC

Return values

The following table lists all the values *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0016002C	Axis ID not accepted
0x0016002B	Axis ID not found
0x0016002D	Axis ID referred to a gantry
0x00160027	Axis ID referred to a spindle
0x0016002A	Axes with different clocks
0x00160146	Maximum number of cams reached
0x00160140	File opening error
0x00160141	LF not found at the end of the file
0x00160142	File non esistente
0x00160143	Table not strictly monotone
0x00160144	First string format wrong
0x00160145	String format wrong
0x00320046	Option A41 disabled

Example

```
Ret      : dword;
CamId   : int ;
FileName : string := '\SSD\OSAI\Camma.csv';

0;9
0;0
100;100
200;200
300;300
400;400
500;300
600;200
700;100
800;0

(*Creates an electronic cam between the master axis with Id1 and the slave axis
with Id 2 *)
Ret := GMC_CamAxiDefine(InterpId,1,CAM_MSTINT,2,FileName, CamId);
(*Creates an electronic cam between the MD variable with Id 100 and the slave
axis with Id 2*)
Ret := GMC_CamAxiDefine(InterpId,100,CAM_MSTVAR,2,FileName, CamId);
```

See also

[GMC_CamAxiStart](#), [GMC_CamAxiStartOnPosition](#), [GMC_CamAxiStartOnSignal](#), [GMC_CamStop](#),
[GMC_CamStopOnPosition](#), [GMC_CamStopOnSignal](#), [GMC_CamUndefine](#), [GMC_ReadAxisInfo](#)

3.101 GMC_CamAxiStart

The function GMC_CamAxiStart enables one or more electronic cams master/slave type.

Syntax

```
ret_code :=GMC_CamAxiStart(InterpId ,OffsetMaster, Mode, Rollover, OffsetSlave, CamStr);
```

Input parameters

InterpId(INT) PLC interpolator identifier [1..40]
OffsetMaster(LREAL) Translation value (early or late) on the application of the slave position
Mode(INT) Tracking mode
Rollover(INT) Cyclic cam modulus
OffsetSlave(LREAL) Translation value applied to the slave position

Input/output parameter

CamStr (Cam_Struct) Structure containing the id of the cam to enables and the action id to which is associated after the activation

Possible overloads

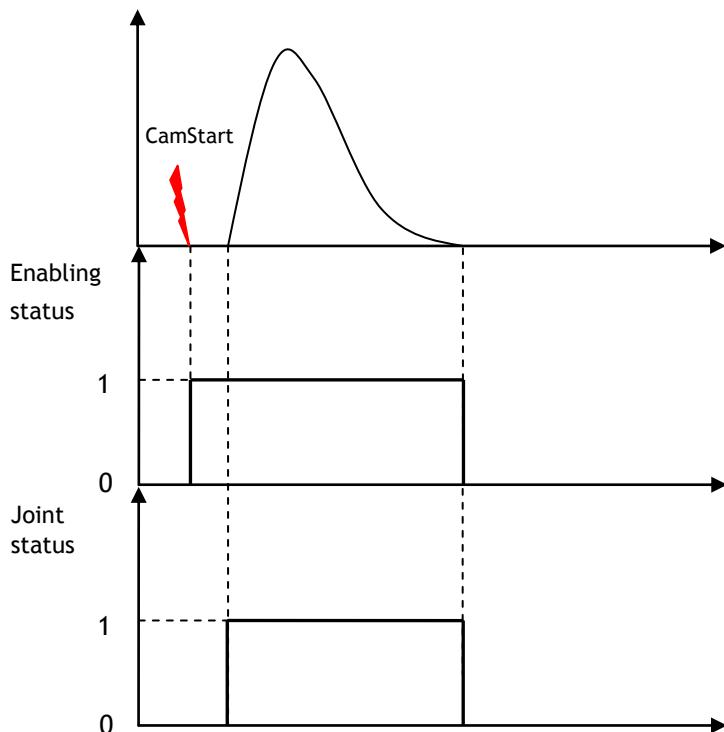
GMC_CamAxiStart(InterpId ,OffsetMaster, Mode, Rollover,[OffsetSlave, CamStr]...);

Execution mode

Wait

Use

The function GMC_CamAxiStart enables the electronic cams in input associating an “ActionId” to them. The enabling is immediate, but the slave joint depends on the master position. Figure below better explains that concept:



The real-time software searches in table the master position and, in case it finds it, calculates the point to pass to the slave. During the activation, the slave could not be close to the first point of the cam profile. Therefore, the slave should move at the best of its dynamic features to reach the position required. According to the input tracking mode, the error between the real position and the theoretical one could be recovered by the axis during the movement going, if possible, faster than the feed programmed.

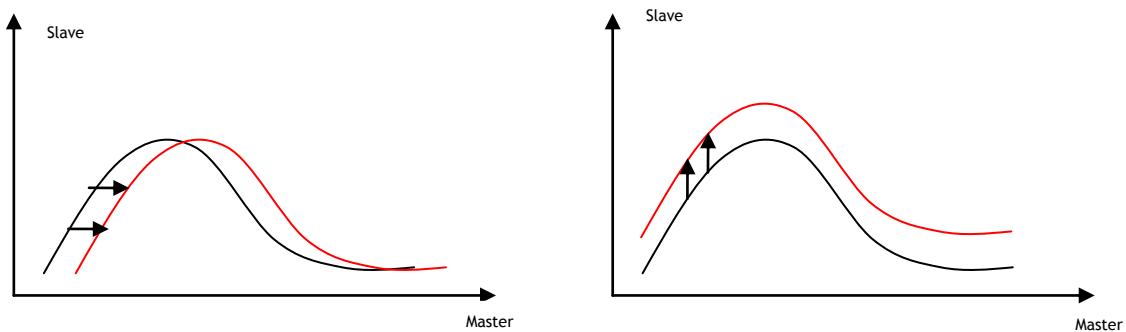
The positions of engagement and disengagement are, respectively, the first and the last point of the table of cam.

The position of the master searched in the table is given by the relation below:

$$Pm = Pm_{att} + OffsetMaster$$

where Pm_{att} is the position of the master including origins (in case of interpolated axis).

Figures below shows the effect of the *OffsetMaster* and *OffsetSlave* application to the cam profile:



Description of the *Cam_Struct* structure:

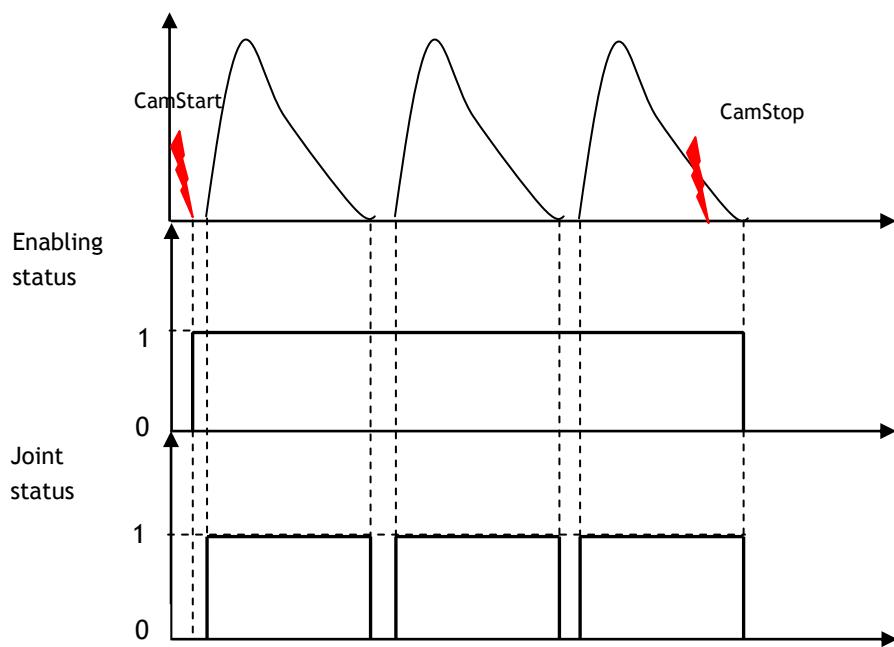
Field	Type	Description
CamID	INT	Identifier of the cam to enable
ActionId	INT	Identifier of the "Action" associated to the cam enabled [1...128]

Description of the field *Mode*:

Mode	Mnemonic	Description
0	CAM_INT	Synchronous tracking (interpolated)
1	CAM_AXF	axf type tracking

The input *Rollover* determines if the cam has to be done once (*Rollover*=0) or infinity (*Rollover* = cyclic cam modulus) until an explicit stop command..

Figure below shows an exemple of cyclic cam:



! Warning It is necessary to use an interpolator PLC in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC

Return values

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160147	Maximum number of Action reached
0x00160148	Invalid cam Id
0x0016014A	Cam nor defined
0x0016014B	Invalid cam cycling modulus(in case of real rollover axis with pitch lower than the cam modulus)
0x0016014D	Release position higher than the cyclic cam modulus
0x00160028	Axis in use by another interpolator
0x00160029	Axis in use by another interpolator in Hold

Example

```

Ret      : dword;
CamId   : int ;
CamStr1 : Cam_Struct;
CamStr2 : Cam_Struct;

(*Enables in one-shot mode the electronic cams with Id 1 and 2 with null
OffsetMaster and null OffsetSlave for the first cam and 10.0 for the second.*)
CamStr1.CamId := 1;
CamStr2.CamId := 2;
Ret := GMC_CamAxiStart(InterpId, 0.0, 0, 0.0, 0.0, CamStr1, 10.0, CamStr2);

```

See also

GMC_CamAxiDefine, GMC_CamAxiStartOnPosition, GMC_CamAxiStartOnSignal, GMC_CamStop, GMC_CamStopOnPosition, GMC_CamStopOnSignal, GMC_CamUndefine, GMC_ReadAxisInfo

3.102 GMC_CamAxiStartOnPosition

The function `GMC_CamAxiStartOnPosition` enables one or more electronic cams slave/master type, according to the value of the `AxisId` axis position. .

Syntax

```
ret_code := GMC_CamAxiStartOnPosition(InterpId, OffsetMaster, Mode, Rollover, AxisId, StartPos,
                                      StartMode, OffsetSlave, CamStr);
```

Input parameters

<code>InterpId(INT)</code>	PLC interpolator identifier [1..40]
<code>OffsetMaster(LREAL)</code>	Translation value (early or late) on the application of the slave position
<code>Mode(INT)</code>	Tracking mode
<code>AxisId(INT)</code>	Axis Id to enable cams
<code>StartPos(LREAL)</code>	Axis position to start the cam
<code>StartMode(INT)</code>	Start mode activation
<code>Rollover(INT)</code>	Cyclic cam modulus
<code>OffsetSlave(LREAL)</code>	Translation value applied to the slave position

Input/output parameter

`CamStr (Cam_Struct)` Structure containing the id of the cam to enables and the action id to which is associated after the activation

Possibili overload

`GMC_CamAxiStartOnPosition (InterpId, OffsetMaster, Mode, Rollover, AxisId, StartPos, StartMode, [OffsetSlave, CamStr]...);`

Execution mode

Wait

Use

The function `GMC_CamAxiStartOnPosition` enables the electronic cams in input according to the axis position.

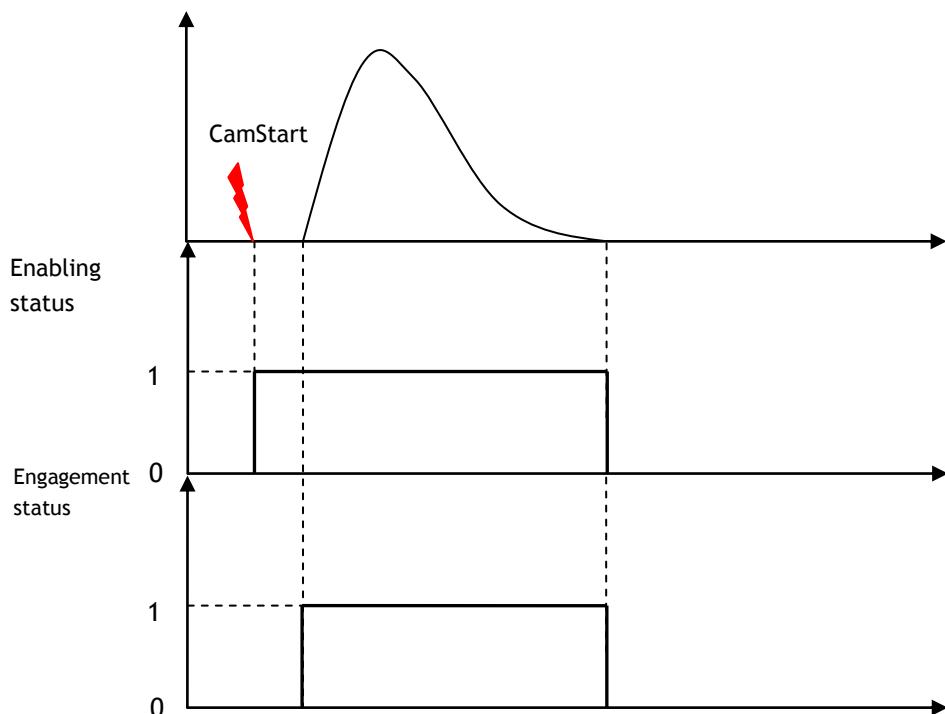
The command is executed when the axis `AxisId` meets the condition `StartMode` respect to the position `StartPos` namely:

<code>StartMode</code>	<code>Mnemonic</code>	<code>Description</code>
10	CND_AXL	Start when the axis <code>AxisId</code> is on an interpolated position lower than <code>StartPos</code> . The position has to take into account the active offsets on the <code>AxisID</code> axis.
11	CND_AXG	Start when the axis <code>AxisId</code> is on an interpolated position higher or equal to <code>StartPos</code> . The position has to take into account the active offsets on the <code>AxisID</code> axis.
12	CND_AXLABS	Start when the axis <code>AxisId</code> is on an interpolated position lower than <code>StartPos</code> . The position is absolute.
13	CND_AXGABS	Start when the axis <code>AxisId</code> is on an interpolated position higher or equal to <code>StartPos</code> . The position is absolute.

14	CND_AXLph	Start when the axis AxisId is on a real position lower hanStartPos. The position has to take into account the active offsets on the AxisID axis.
15	CND_AXGph	Start when the axis AxisId is on a real position higher or equal to StartPos. The position has to take into account the active offsets on the AxisID axis.
16	CND_AXLABSph	Start when the axis AxisId is on a real position lower than StartPos. The position is absolute.
17	CND_AXGABSpj	Start when the axis AxisId is on a real position higher or equal to StartPos. The position is absolute.

The function finishes when the command is executed. The absolute/incremental programming mode is NOT used from the StartPos parameter, therefore the position is considered as absolute; the origins applied to the axis are taken into account anyway. The enabling is immediate, but the slave joint depends on the master position.

Figure below better explains this concept:



The real-time software searches in table the master position and, in case it finds it, calculates the point to pass to the slave. During the activation, the slave could not be close to the first point of the cam profile. Therefore, the slave should move at the best of its dynamic features to reach the position required. According to the input tracking mode, the error between the real position and the theoretical one could be recovered by the axis during the movement going, if possible, faster than the feed programmed.

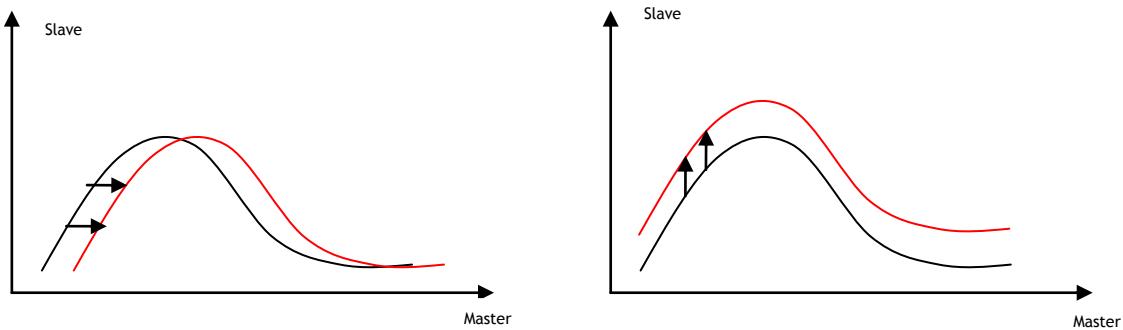
The positions of engagement and disengagement are, respectively, the first and the last point of the table of cam.

The position of the master searched in the table is given by the following relation

$$Pm = Pm_{att} + OffsetMaster$$

where Pm_{att} is the position of the master including origins in case of interpolated axis.

Figures below show the effect of the application $OffsetMaster$ and $OffsetSlave$ on the cam profile:



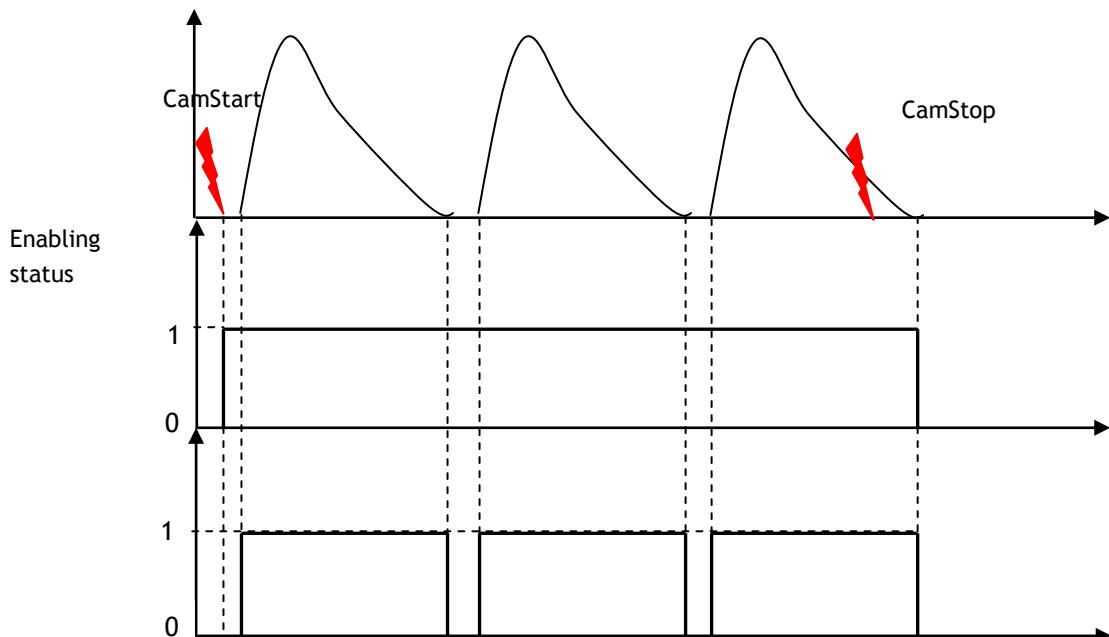
Description of the *Cam_Struct* structure:

Field	Type	Description
CamID	INT	Identifier of the cam to enable
ActionId	INT	Identifier of the "Action" associated to the cam enabled [1...128]

Description of the field *Mode*:

Mode	Mnemonic	Description
0	CAM_INT	Synchronous tracking (interpolated)
1	CAM_AXF	axf type tracking

The input *Rollover* determines if the cam has to be done once (*Rollover*=0) or infinity (*Rollover* = cyclic cam modulus) until an explicit stop command. The figure below shows an example of cyclic cam:



It is necessary to use an interpolator PLC in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC

Return values

The following table lists all values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160147	Maximum number of Action reached
0x00160148	Invalid cam Id
0x0016014A	Cam nor defined
0x0016014B	Invalid cam cycling modulus(in case of real rollover axis with pitch lower than the cam modulus)
0x0016014D	Release position higher than the cyclic cam modulus
0x00160028	Axis in use by another interpolator
0x00160029	Axis in use by another interpolator in Hold
0x00160032	Wrong start movement condition

Example

```

Ret      : dword;
CamId   : int ;
CamStr1 : Cam_Struct;
CamStr2 : Cam_Struct;
CamStr1.CamId := 1;
CamStr2.CamId := 2;
(*Enables in one-shot mode the electronic cams with Id 1 and 2 when the axis 1
has a position interpolated equal or higher than 100*)
Ret := GMC_CamAxiStartOnPosition(InterpId, 0.0, 0, 0.0, 1, 100.0, CND_AXG, 0.0,
CamStr1, 10.0, CamStr2);

```

See also

[GMC_CamAxiDefine](#), [GMC_CamAxiStart](#), [GMC_CamAxiStartOnSignal](#), [GMC_CamStop](#),
[GMC_CamStopOnPosition](#), [GMC_CamStopOnSignal](#), [GMC_CamUndefine](#), [GMC_ReadAxisInfo](#)

3.103 GMC_CamAxStartOnSignal

The function `GMC_CamAxStartOnSignal` enables one or more electronic cams master/slave type on signal.

Syntax

```
ret_code := GMC_CamAxStartOnSignal(InterId, OffsetMaster, Mode, Rollover, StartSignal, StartMode,
OffsetSlave, CamStr);
```

Input parameters

<code>InterId(INT)</code>	PLC interpolator identifier [1..40]
<code>OffsetMaster(LREAL)</code>	Translation value(early or late) on the application of the slave position
<code>Mode(INT)</code>	Tracking mode
<code>AxisId(INT)</code>	Axis Id to enable cams
<code>StartSignal(BOOL)</code>	Boolean variable to enable cam
<code>StartMode(INT)</code>	Start mode of the activation
<code>Rollover(INT)</code>	Cyclic cam modulus
<code>OffsetSlave(LREAL)</code>	Translation value applied to the slave position

Input/output parameter

`CamStr (Cam_Struct)` Structure containing the id of the cam to enable and the id of the action to which is associated after the activation

Possible overload

`GMC_CamAxStartOnSignal(InterId, OffsetMaster, Mode, Rollover, AxisId, StartPos, StartMode, [OffsetSlave, CamStr]...);`

Execution mode

Wait

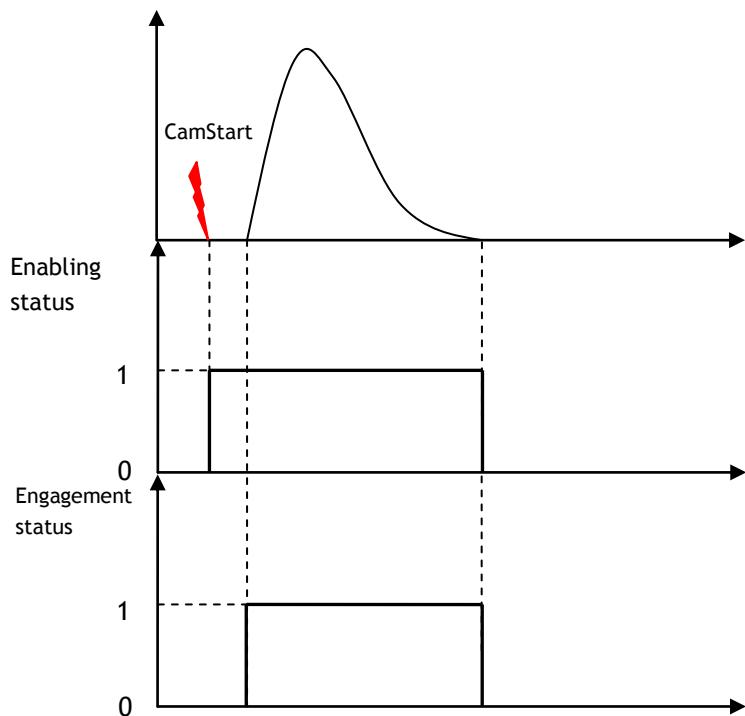
Use

The function `GMC_CamAxStartOnSignal` enables the electronic cams in inputs according to a signal status. The command is executed when the signal `StartSignal` complies the condition `StartMode` that is:

StartMode	Mnemonic	Description
1	CND_RAISE	Start on the rising edge of the signal Signal
2	CND_FAIL	Start on the falling edge of the signal Signal
3	CND_ON	Start on level ON (true) of the signal Signal
4	CND_OFF	Start on level OFF (false) of the signal Signal

The function finishes when the command is executed. The activation is immediate, but the slave joint depends on the master position.

Figure below better explains this concept:



The real-time software searches in table the master position and, in case it finds it, calculates the point to pass to the slave. During the activation, the slave could not be close to the first point of the cam profile. Therefore, the slave should move at the best of its dynamic features to reach the position required. According to the input tracking mode, the error between the real position and the theoretical one could be recovered by the axis during the movement going, if possible, faster than the feed programmed.

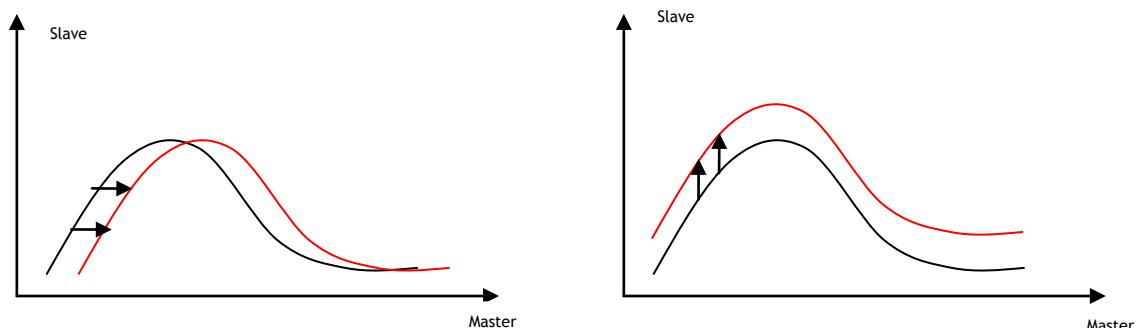
The positions of engagement and disengagement are, respectively, the first and the last point of the table of cam.

The position of the master searched in the table is given by the following relation

$$Pm = Pm_{att} + OffsetMaster$$

where Pm_{att} is the position of the master including origins (in case of interpolated axis).

Figures below show the effect of the *OffsetMaster* and *OffsetSlave* application on the cam profile:



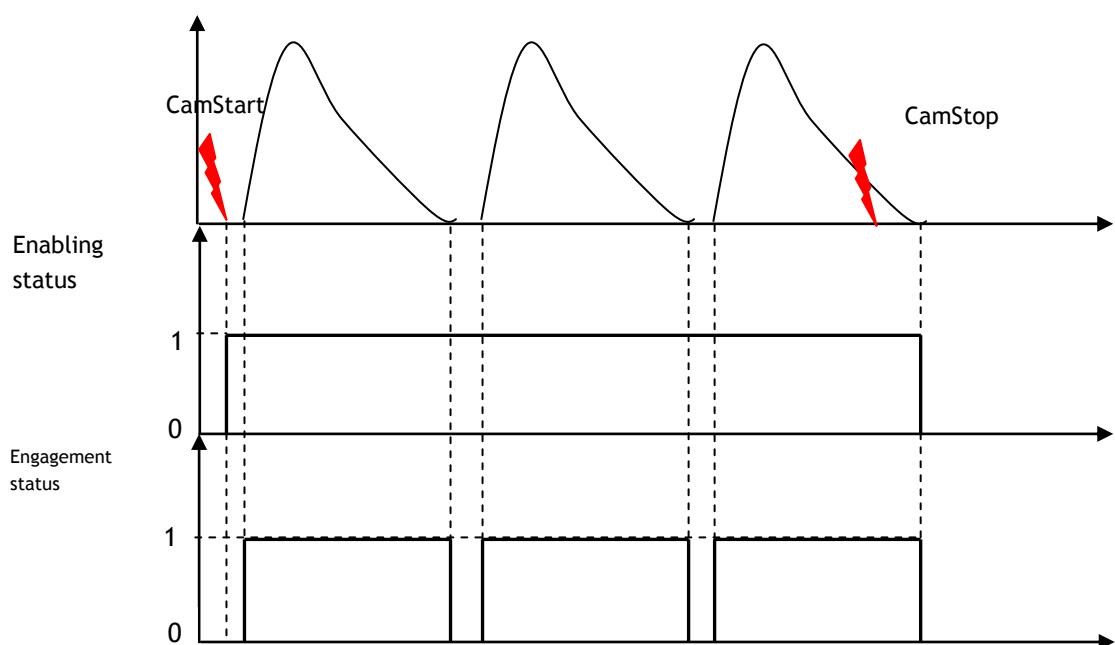
Description of the *Cam_Struct* structure:

Field	Type	Description
CamID	INT	Identifier of the cam to enable
ActionId	INT	Identifier of the “Action” associated to the cam enabled [1...128]

Description of the field *Mode*:

Mode	Mnemonic	Description
0	CAM_INT	Synchronous tracking (interpolated)
1	CAM_AXF	axf type tracking

The input *Rollover* determines if the cam has to be done once (*Rollover*=0) or infinity (*Rollover* = cyclic cam modulus) until an explicit stop command. The figure below shows an example of cyclic cam:



It is necessary to use an interpolator PLC in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC

Return values

The following table lists all values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Descrizione
0x00000000	Function executed without errors
0x00160147	Maximum number of Action reached
0x00160148	Invalid cam Id
0x0016014A	Cam not defined
0x0016014B	Invalid cam cycling modulus (in case of real rollover axis with pitch lower than the cam modulus)
0x0016014D	Release position higher than the cyclic cam modulus
0x00160028	Axis in use by another interpolator
0x00160029	Axis in use by another interpolator in Hold
0x00160032	Wrong start movement condition
0x00160036	Signal synchronism variable wrong

Example

```
Ret      : dword;
CamId    : int ;
CamStr1  : Cam_Struct;
CamStr2  : Cam_Struct;

(*Enables in one-shot mode the electronic cams with Id 1 and on the rising edge
of the signal M0_0*)

CamStr1.CamId := 1;
CamStr2.CamId := 2;
Ret := GMC_CamAxiStartOnSignal(InterpId, 0.0, 0, 0.0, M0_0, CND_RAISE, 0.0,
CamStr1, 10.0, CamStr2);
```

See also

[GMC_CamAxiDefine](#), [GMC_CamAxiStart](#), [GMC_CamAxiStartOnPosition](#), [GMC_CamStop](#),
[GMC_CamStopOnPosition](#), [GMC_CamStopOnSignal](#), [GMC_CamUndefine](#), [GMC_ReadAxisInfo](#)

3.104 GMC_CamOutDefine

The function GMC_CamOutDefine defines an electronic cam associated to a master to the outputs.

Syntax

```
ret_code := GMC_CamOutDefine (InterId, MasterId, MstType ,FileName, out CamId) ;
```

Input parameters

<i>InterId</i> (INT)	PLC interpolated identifier [1..40]
<i>MasterId</i> (INT)	Master identifier
<i>MstType</i> (INT)	Master type
<i>FileName</i> (STRING)	String containing the correspondence between master and output

Output parameter

<i>CamId</i> (INT)	Electronic cam identifier [1..64]
--------------------	-----------------------------------

Execution mode

Wait

Use

The function GMC_CamOutDefine creates an association between a master and a series of outputs that are set/reset in correspondence to master positions. This association is preset in tables.

MasterId is the identifier of the master that can be an interpolated axis (physical or virtual), a transducer or a MD variable. The field *MstType* can therefore have the following values:

<i>MstType</i>	<i>Mnemonic</i>	Description
0	CAM_MSTINT	Interpolated axis
1	CAM_MSTTRAS	Transducer
2	CAM_MSTVAR	MD variable

Tables are created from a .csv file. The file is organized by lines, therefore each line refers to a cam position. The first line contains information referring to the file type and to the number of strings compiled. Other strings contain the correspondence between the master position and the outputs to set/reset.

First string →	2;2	FileType; Number of strings
Other strings →	100;OW(13,61440,65280)	Master;OW(index,value,mask) <i>Output standard</i>
	200;OW(14,0xF000,0xFF00)	
	300;OF(0x01, 0x03)	Master;OF(value,mask) <i>Fast Output</i>
	400; MW(15,0xF000,0xFF00)	Master;MW(index,value,mask) <i>Variabili MW</i>

When the cam is enabled, it searches the master position within the table and the outputs associated to the corresponding word are set/reset. The master axis positions have to be increasing or decreasing monotone.



It is necessary to use an interpolator PLC in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC

Return values

The following table lists all the values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0016002C	Axis ID not accepted
0x0016002B	Axis ID not found
0x0016002D	Axis ID referred to a gantry
0x00160027	Axis ID referred to a spindle
0x0016002A	Axes with different clock
0x00160146	Maximum number of cams reached
0x00160140	File opening error
0x00160141	LF not found at the end of the file
0x00160142	Non-existent file
0x00160143	Table not monotone
0x00160144	Wrong first string format
0x00160145	Wrong string format
0x00320046	Option A41 disabled

Example

```

Ret      : dword;
CamId   : int ;
FileName : string := '\SSD\OSAI\CammaOUT.csv';

0;9
2;12
10;OW(13,61440,65280)
20;OW(14,0xF000,0xFF00)
30;OW(13,3840,65280)
40;OW(14,0x0F00,0xFF00)
50;OW(13,61440,65280)
60;OW(14,0xF000,0xFF00)
70;OW(13,3840,65280)
80;OW(14,0x0F00,0xFF00)
90;OW(13,61440,65280)
100;OW(14,0xF000,0xFF00)
110;OW(13,3840,65280)
120;OW(14,0x0F00,0xFF00)

(*Creates an electronic cam managing some outputs *)
Ret := GMC_CamOutDefine(InterpId,1,CAM_MSTINT, FileName, CamId);
(*Creates an electronic cam with the variable MD with Id 100 as master*)
Ret := GMC_CamOutDefine (InterpId,100,CAM_MSTVAR, FileName, CamId);

```

See also

GMC_CamOutStart, GMC_CamOutStartOnPosition, GMC_CamOutStartOnSignal, GMC_CamStop, GMC_CamStopOnPosition, GMC_CamStopOnSignal, GMC_CamUndefine, GMC_ReadAxisInfo

3.105 GMC_CamOutStart

The function GMC_CamOutStart enables an electronic cam output type.

Syntax

```
ret_code := GMC_CamOutStart(InterpId, CamID, OffsetMaster, Rollover, ActionID);
```

Input parameters

<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>CamID</i> (INT)	Identifier of the cam to enable [1..64]
<i>OffsetMaster</i> (LREAL)	Translation value (early or late) on outputs set/reste
<i>Rollover</i> (INT)	Cyclic cam modulus

Output parameter

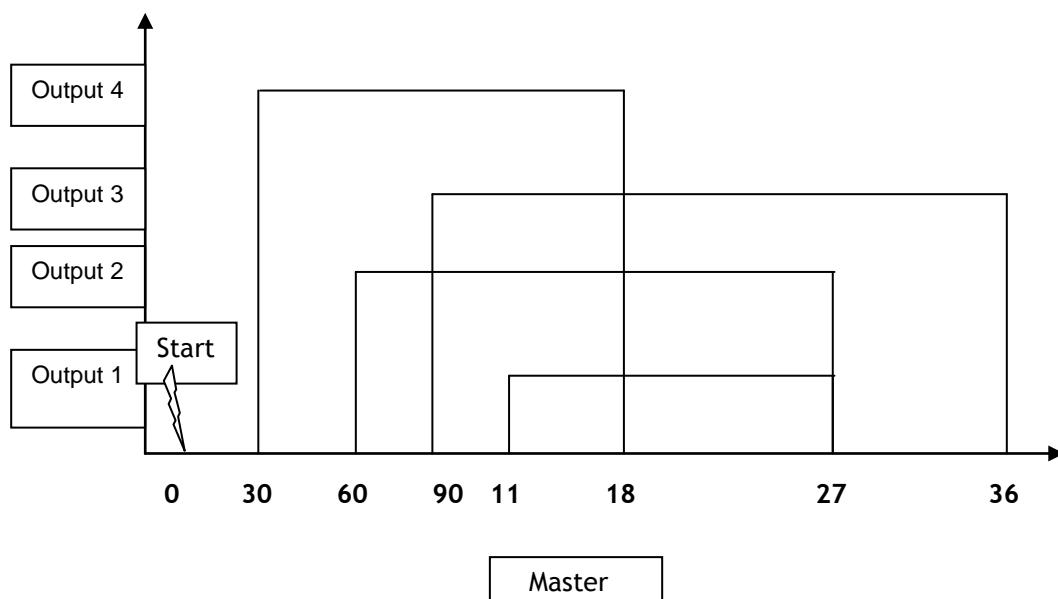
<i>ActionID</i> (INT)	Created action identifier [1..128]
-----------------------	------------------------------------

Execution mode

Wait

Use

The function GMC_CamOutStart enables the electronic cam in input associating it tot an “ActionId”. The enabling is immediate, but the outputs management depends on the master position.



The real-time software searches within the table the master position and, if found, manages the corresponding outputs.

The master position searched in table is given by the following relation

$$Pm = Pm_{att} + OffsetMaster$$

where Pm_{att} the master position includes origins in case of an interpolated axis.

The *Rollover* input sets if the cam has to be tracked once (Rollover=0) or infinity (Rollover = cyclic cam modulus) until a stop command is given.



It is necessary to use a PLC interpolator in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be any axes in the *OPENcontrol* system currently associated with the PLC

Return values

The following table lists all the values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160147	Maximum number of Action reached
0x00160148	Invalid cam ID
0x0016014A	Cam not defined
0x0016014B	Invalid cyclic cam modulus (in case of real rollover with pitch lower than the cam modulus)
0x0016014D	Release position higher than the cyclic cam modulus

Example

```
Ret      : dword;
CamId   : int ;
ActionId : int ;
(*Enables in cyclic mode the electronic cam with Id = 1*)
CamId := 1;
Ret := GMC_CamOutStart(InterpId, 1, 360.0, 0, ActionId);
```

See also

[GMC_CamOutDefine](#), [GMC_CamOutStartOnPosition](#), [GMC_CamOutStartOnSignal](#), [GMC_CamStop](#), [GMC_CamStopOnPosition](#), [GMC_CamStopOnSignal](#), [GMC_CamUndefine](#), [GMC_ReadAxisInfo](#)

3.106 GMC_CamOutStartOnPosition

The function GMC_CamOutStartOnPosition enables an electronic cam output type.

Syntax

```
ret_code := GMC_CamOutStartOnPosition(InterpId, CamID, OffsetMaster, Rollover, AxisId, StartPos,
                                      StartMode, ActionID);
```

Input parameters

<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>CamID</i> (INT)	Identifier of the cam to enable [1..64]
<i>OffsetMaster</i> (LREAL)	Translation value (early or late) on outputs set/reset
<i>Rollover</i> (INT)	Cyclic cam modulus
<i>AxisId</i> (INT)	Axis id to enable the cams
<i>StartPos</i> (LREAL)	Axis position to enable the cam
<i>StartMode</i> (INT)	Activation start mode

Output parameter

<i>ActionID</i> (INT)	Created action identifier [1..128]
-----------------------	------------------------------------

Execution mode

Wait

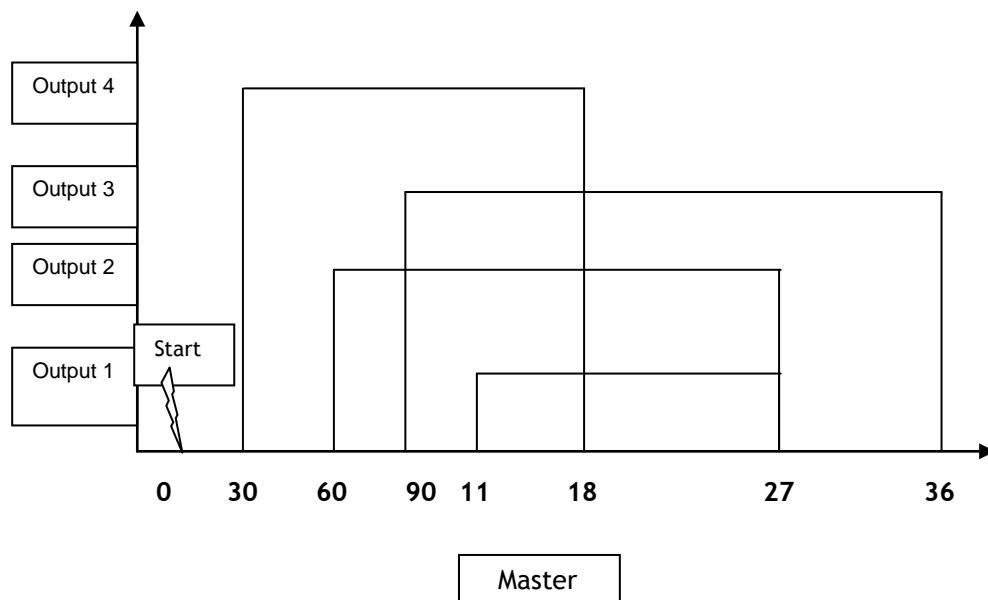
Use

The function GMC_CamOutStartOnPosition enables the electronic cams in input according to the axis position. The command is executed when the axis *AxisId* meets the condition *StartMode* respect to the position *StartPos* namely:

<i>StartMode</i>	<i>Mnemonic</i>	<i>Description</i>
10	CND_AXL	Start when the axis <i>AxisId</i> is on an interpolated position lower than <i>StartPos</i> . The position has to take into account the active offsets on the <i>AxisID</i> axis.
11	CND_AXG	Start when the axis <i>AxisId</i> is on an interpolated position higher or equal to <i>StartPos</i> . The position has to take into account the active offsets on the <i>AxisID</i> axis.
12	CND_AXLABS	Start when the axis <i>AxisId</i> is on an interpolated position lower than <i>StartPos</i> . The position is absolute.
13	CND_AXGABS	Start when the axis <i>AxisId</i> is on an interpolated position higher or equal to <i>StartPos</i> . The position is absolute.
14	CND_AXLph	Start when the axis <i>AxisId</i> is on a real position lower han <i>StartPos</i> . The position has to take into account the active offsets on the <i>AxisID</i> axis.
15	CND_AXGph	Start when the axis <i>AxisId</i> is on a real position higher or equal to <i>StartPos</i> . The position has to take into account the active offsets on the <i>AxisID</i> axis.
16	CND_AXLABSp	Start when the axis <i>AxisId</i> is on a real position lower than <i>StartPos</i> . The position is absolute.
17	CND_AXGABSpj	Start when the axis <i>AxisId</i> is on a real position higher or equal to <i>StartPos</i> . The position is absolute.

The function finishes when the command is executed. The absolute/incremental programming mode is NOT used for the StartPos parameter, therefore the position is considered only as absolute; the origins applied to the axis are taken into account anyway.

The activation is immediate, but the management of the outputs depends on the master position.



The real-time software searches within the table the master position and, if found, manages the corresponding outputs.

The master position searched in table is given by the following relation

$$Pm = Pm_{att} + OffsetMaster$$

where Pm_{att} the master position includes origins in case of an interpolated axis.

The *Rollover* input sets if the cam has to be tracked once (*Rollover=0*) or infinity (*Rollover = cyclic cam modulus*) until a stop command is given.



It is necessary to use a PLC interpolator in order to execute commands for managing Cam axes, however, the axes cannot belong to the interpolator, but may be any axes in the OPENcontrol system currently associated with the PLC.

Return values

<i>ret_code (HEX)</i>	<i>Description</i>
0x00000000	Function executed without errors
0x00160147	Maximum number of Action reached
0x00160148	Invalid cam ID
0x0016014A	Cam not defined
0x0016014B	Invalid cyclic cam modulus (in case of real rollover with pitch lower than the cam modulus)
0x0016014D	Release position higher than the cyclic cam modulus
0x00160028	Axis used by another interpolator
0x00160029	Axis used by another interpolator in Hold
0x00160032	Wrong conditions of string synchronization

Example

```
Ret      : dword;
CamId    : int ;
ActionId : int ;
CamStr1  : Cam_Struct;
CamStr2  : Cam_Struct;
CamId := 1;
(*Enables in cyclic mode the electronic cam with Id 1 when the axis 1 has a an
interpolated position equal or higher than 100*)
Ret := GMC_CamOutStartOnPosition(InterpId, 1, 0.0, 360.0, 1, 100.0, CND_AXG,
ActionId);
```

See also

[GMC_CamOutDefine](#), [GMC_CamOutStart](#), [GMC_CamOutStartOnSignal](#), [GMC_CamStop](#),
[GMC_CamStopOnPosition](#), [GMC_CamStopOnSignal](#), [GMC_CamUndefine](#), [GMC_ReadAxisInfo](#)

3.107 GMC_CamOutStartOnSignal

La funzione GMC_CamOutStartOnSignal attiva una o più camme elettroniche di tipo master/slave su segnale.

Sintassi

```
ret_code := GMC_CamOutStartOnSignal (InterpId, CamId, OffsetMaster, Rollover, StartSignal,
                                     StartMode, ActionID);
```

Input parameters

<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>CamID</i> (INT)	Identifier of the cam to enable [1..64]
<i>OffsetMaster</i> (LREAL)	Translation value (early or late) on outputs set/reset
<i>Rollover</i> (INT)	Cyclic cam modulus
<i>StartSignal</i> (BOOL)	Boolean variable to enable the cam
<i>StartMode</i> (INT)	Activation start mode

Output parameter

<i>ActionID</i> (INT)	Created action identifier [1..128]
-----------------------	------------------------------------

Execution mode

Wait

Use

The function GMC_CamOutStartOnSignal enables the electronic cams in input according to a signal status. The command is executed when the signal *StartSignal* complies the condition *StartMode* that is:

<i>StartMode</i>	<i>Mnemonic</i>	<i>Description</i>
1	CND_RAISE	Start on the rising edge of the signal Signal
2	CND_FAIL	Start on the falling edge of the signal Signal
3	CND_ON	Start on level ON (true) of the signal Signal
4	CND_OFF	Start on level OFF (false) of the signal Signal

The function finishes when the command is executed. The activation is immediate, but the outputs management depends on the master positon.

The real-time software searches within the table the master position and, if found, manages the corresponding outputs.

The master position searched in table is given by the following relation

$$Pm = Pm_{att} + OffsetMaster$$

where Pm_{att} the master position includes origins in case of an interpolated axis.



It is necessary to use an interpolator PLC in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC

Return values

The following table lists the values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160147	Maximum number of Action reached
0x00160148	Invalid cam ID
0x0016014A	Cam not defined
0x0016014B	Invalid cyclic cam modulus (in case of real rollover with pitch lower than the cam modulus)
0x0016014D	Release position higher than the cyclic cam modulus
0x00160028	Axis used by another interpolator
0x00160029	Axis used by another interpolator in Hold
0x00160032	Wrong conditions of strting synchronism
0x00160036	Signal synchronism variable wrong

Example

```
Ret      : dword;
CamId   : int ;
ActionId : int ;
(*Enables in cyclic mode the electronic cam with Id1 on rising edge of M0_0*)
CamId := 1;
Ret := GMC_CamOutStartOnSignal(InterpId, 1, 0.0, 360.0, M0_0, CND_RAISE,
ActionId);
```

See also

[GMC_CamOutDefine](#), [GMC_CamOutStart](#), [GMC_CamOutStartOnPosition](#), [GMC_CamStop](#),
[GMC_CamStopOnPosition](#), [GMC_CamStopOnSignal](#), [GMC_CamUndefine](#), [GMC_ReadAxisInfo](#)

3.108 GMC_CamStop

The function GMC_CamStop disables one or more active cams.

Syntax

```
ret_code := GMC_CamStop(InterpId , Mode, ActionId);
```

Input parameters

<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>Mode</i> (INT)	Interruption mode
<i>ActionId</i> (INT)	Identifier of the action to stop [1..128]

Possible overload

GMC_CamStop(*InterpId*, *Mode*, [*ActionId*]...);

Execution mode

Wait

Use

The function GMC_CamStop immediately disables the input action. The *Mode* input gets different meanings according to the type of cam to stop.

<i>Mode</i>	<i>Mnemonic</i>	Master/Slave cam	Output cam
0	CAM_STOP0	The slave axis immediately stops with its deceleration ramp and the action in input is deleted.	Deletes the current action without changing the outputs status
1	CAM_STOP1	The slave axis only stops once the profile is completed. At the end of the movement the action is deleted. This command is redundant for one-shot cams as the actions are automatically deleted at the end of the profile execution.	Deletes the current action after resetting all the outputs



It is necessary to use an interpolator PLC in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC

Return values

The following table lists the values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160149	Invalid action ID

Example

```
Ret      : dword;
ActionId : int := 1;
(*Immediately disables the electronic cam associated to the action with Id = 1*)
Ret := GMC_CamStop(InterpId, 0, ActionId);
```

See also

GMC_CamAxiDefine, GMC_CamOutDefine, GMC_CamAxiStart, GMC_CamAxiStartOnPosition,
GMC_CamAxiStartOnSignal, GMC_CamOutStart, GMC_CamOutStartOnPosition,
GMC_CamOutStartOnSignal, , GMC_CamStopOnPosition, GMC_CamStopOnSignal, GMC_CamUndefine,
GMC_ReadAxisInfo

3.109 GMC_CamStopOnSignal

The function GMC_CamStopOnSignal disables one or more cams active on signal.

Syntax

```
ret_code := GMC_CamStopOnSignal(InterpId, Mode, StopSignal, StopSignMode, ActionId);
```

Input parameters

<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>Mode</i> (INT)	Interruption mode
<i>StopSignal</i> (BOOL)	Boolean variable to disable the cam
<i>StopSignMode</i> (INT)	Command of the start modality
<i>ActionId</i> (INT)	Identifier of the action to stop [1..128]

Possible overload

GMC_CamStop(*InterpId*, *Mode*, [*ActionId*]...);

Execution mode

Wait

Possible overload

GMC_CamStop(*InterpId*, *Mode*, *StopSignal*, *StopSignMode*, [*ActionId*]...);

Execution mode

Wait

Use

The function GMC_CamStopOnSignal disables the action in input according to a signal status. The command is executed when the signal *StopSignal* complies with the condition *StopSignMode* that is:

StartMode	Mnemonic	Description
1	CND_RAISE	Start on rising edge of signal Signal
2	CND_FAIL	Start on falling edge of signal Signal
3	CND_ON	Start on “ON” level (true) of signal Signal
4	CND_OFF	Start on “OFF” level (false) of signal Signal

The input *Mode* can get different meanings according to the type of cam to stop.

Mode	Mnemonic	Master/Slave cam	Output cam
0	CAM_STOP0	The slave axis immediately stops with its deceleration ramp and the action in input is deleted.	Deletes the current action without changing the outputs status
1	CAM_STOP1	The slave axis only stops once the profile is completed. At the end of the movement the action is deleted. This command is redundant for one-shot cams as the actions are automatically deleted at the end of the profile execution.	Deletes the current action after resetting all the outputs



It is necessary to use an interpolator PLC in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC

Return values

The following table lists the values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160149	Invalid action ID
0x00160032	Wrong start command condition
0x00160036	StopSignal synchronism variable wrong

Example

```
Ret      : dword;
ActionId : int := 1;
(*Disables the electronic cam associated to the action Id = 1 on the rising
edge of M0_0*)
Ret := GMC_CamStop(InterpId, 0, M0_0, CND_RAISE, ActionId);
```

See also

GMC_CamAxiDefine, GMC_CamOutDefine, GMC_CamAxiStart, GMC_CamAxiStartOnPosition,
 GMC_CamAxiStartOnSignal, GMC_CamOutStart, GMC_CamOutStartOnPosition,
 GMC_CamOutStartOnSignal, GMC_CamStopOnPosition, GMC_CamStop, GMC_CamUndefine,
 GMC_ReadAxisInfo

3.110 GMC_CamStopOnPosition

The function GMC_CamStopOnPosition disables one or more active cams according tot an axis position.

Syntax

```
ret_code := GMC_CamStopOnPosition(InterpId , Mode, AxisId, StopPos, StopPosMode ActionId);
```

Input parameters

<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>Mode</i> (INT)	Interruption mode
<i>AxisId</i> (INT)	Axis id to disable cams
<i>StopPos</i> (LREAL)	Axis position to disable cams
<i>StopPosMode</i> (INT)	Modality of the command start
<i>ActionId</i> (INT)	Identifier of the action to stop [1..128]

Possible overload

GMC_CamStop(*InterpId*, *Mode*, *StopSignal*, *StopSignMode*, [*ActionId*]...);

Execution mode

Wait

Use

The function GMC_CamStop disabòles the action in input according to an axis position. The command is executed when the axis *AxisId* complies with the condition *StopPosMode* respect to the *StopPos* that is:

<i>StartMode</i>	<i>Mnemonic</i>	<i>Description</i>
10	CND_AXL	Start when the axis AxisId is on an interpolated position lower than StartPos. The position has to take into account the active offsets on the AxisID axis.
11	CND_AXG	Start when the axis AxisId is on an interpolated position higher or equal to StartPos. The position has to take into account the active offsets on the AxisID axis.
12	CND_AXLABS	Start when the axis AxisId is on an interpolated position lower than StartPos. The position is absolute.
13	CND_AXGABS	Start when the axis AxisId is on an interpolated position higher or equal to StartPos. The position is absolute.
14	CND_AXLph	Start when the axis AxisId is on a real position lower hanStartPos. The position has to take into account the active offsets on the AxisID axis.
15	CND_AXGph	Start when the axis AxisId is on a real position higher or equal to StartPos. The position has to take into account the active offsets on the AxisID axis.
16	CND_AXLABSp	Start when the axis AxisId is on a real position lower than StartPos. The position is absolute.
17	CND_AXGABSpj	Start when the axis AxisId is on a real position higher or equal to StartPos. The position is absolute.

The absolute/incremental programming modality is NOT used from the StopPos parameter, therefore the position is only considered as absolute; the origin applied to the axis are taken into account anyway.

The input *Mode* can get different meanings according to the type of cam to stop.

Mode	Mnemonic	Master/Slave cam	Output cam
0	CAM_STOP0	The slave axis immediately stops with its deceleration ramp and the action in input is deleted.	Deletes the current action without changing the outputs status
1	CAM_STOP1	The slave axis only stops once the profile is completed. At the end of the movement the action is deleted. This command is redundant for one-shot cams as the actions are automatically deleted at the end of the profile execution.	Deletes the current action after resetting all the outputs



It is necessary to use an interpolator PLC in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC.

Return values

The following table lists the values *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00160149	Invalid action ID

Example

```
Ret      : dword;
ActionId : int := 1;
(*Disables the electronic cam associated to the action with Id = 1 when the
position of the axis with id = 1 is equal or higher than 100*)
Ret := GMC_CamStop(InterpId, 0, 1, 100.0, CND_AXG, ActionId);
```

See also

GMC_CamAxiDefine, GMC_CamOutDefine, GMC_CamAxiStart, GMC_CamAxiStartOnPosition,
 GMC_CamAxiStartOnSignal, GMC_CamOutStart, GMC_CamOutStartOnPosition, GMC_CamOutStartOn, ,
 GMC_CamStopOnPosition, GMC_CamStopOnSignal, GMC_CamUndefine, GMC_ReadAxisInfo

3.111 GMC_CamUndefine

The function GMC_CamUndefine deletes one or more input cams.

Syntax

```
ret_code := GMC_CamUndefine(InterpId , CamId);
```

Input parameters

<i>InterpId</i> (INT)	PLC interpolator identifier [1..40]
<i>CamId</i> (INT)	Identifier of the cam to delete [1..64]

Possible overload

```
GMC_CamUndefine(InterpId, [CamId]...);
```

Execution mode

Wait

Use

The function GMC_CamUnDefine deletes the input cams.



It is necessary to use a PLC interpolator in order to execute commands for managing Cam axes, however, the axes can not belong to the interpolator specified, but may be all axes in the system XTend currently associated with the PLC.

Return values

The following table lists the values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00160148	Invalid cam ID
0x0016014C	Cam busy in an action

Example

```
Ret      : dword;
CamId   : int := 1;
(*Deletes the electronic cam with Id = 1*)
Ret := GMC_CamUndefine(InterpId, 1);
```

See also

GMC_CamAxiDefine, GMC_CamOutDefine, GMC_CamAxiStart, GMC_CamAxiStartOnPosition,
 GMC_CamAxiStartOnSignal, GMC_CamOutStart, GMC_CamOutStartOnPosition,
 GMC_CamOutStartOnSignal, GMC_CamStopOnPosition, GMC_CamStopOnSignal, GMC_CamStop,
 GMC_ReadAxisInfo

3.112 Motion Control error list

Error code (HEX)	Description
0x00160001	Too many processes/interpolators System is creating too many processes or moves.
0x00160003	Too many interpolation activities System is creating too many interpolations.
0x00160004	To many moves System is creating too many moves compared to the ODM configuration (pre-calculation buffer number).
0x00160005	Wrong axes number Request for moving more than 12 axes at the same time.
0x00160006	Non-existent move The move requested is not available anymore.
0x00160007	Not reserved move The move requested doesn't belong to the process nor to the requiring interpolator.
0x00160008	Non-existent interpolator/process The process or the interpolator is not available anymore.
0x00160009	Memory allocation error When booting the system, memory doesn't have enough space to configure the ODM. Please, contact the customer service.
0x0016000A	Instruction not allowed in point-to-point mode Command is unavailable in point-to-point mode.
0x0016000B	Command not allowed in continuous mode The instruction is not available in continuous mode.
0x0016000C	Program terminated The system is required to perform a move while terminating (reset or emergency).
0x0016000D	Too many interpolators active The system is creating too many move activities (for example, Select and move two axes in manual mode creates two move activities)
0x0016000E	Command not allowed in Abort Continues System is trying to perform a move while it is stopping a continuous movement due to a reset or an emergency.
0x0016000F	Instruction not allowed System is trying to perform a move while it is performing another movement.
0x00160010	Wrong interpolator/process managing positions System is trying to configure a process or an interpolator in an erroneous way. This error does not depend on the ODM process configuration.
0x00160011	Wrong Interpolator/Process Type System is trying to configure a process or an interpolator in an erroneous way. This error does not depend on the ODM process configuration.

Error code (HEX)	Description
0x00160012	Wrong interpolator/process error managing mode System is trying to configure a process or an interpolator in an erroneous way. This error does not depend on the ODM process configuration.
0x00160013	Wrong interpolator/process emergency managing mode System is trying to configure a process or an interpolator in an erroneous way. This error does not depend on the ODM process configuration.
0x00160014	Interpolator/process not found System is trying to execute a movement on a non-existent process or interpolator.
0x00160015	Interpolator/Process already defined System is trying to generate a process or an interpolator that exists already.
0x00160016	Interpolator/Process still working Impossible to close an interpolator or a process while it still owing move activities.
0x00160017	Movement still running Impossible to free a movement while it is still working.
0x00160018	Interpolation start denied The request for a move start as been rejected since the interpolator is currently busy.
0x00160019	Hold mode denied The requested interpolator cannot be put in hold since it is currently in error, emergency or reset state.
0x0016001A	Exit from Hold denied Requested an ‘exit’ from hold for an interpolator that is not in hold state.
0x0016001B	Axes out of profile while exiting Hold During an exit from Hold at least one axis is not on the position where it was when it entered in hold state. The error occurs when trying to restart a suspended movement without setting the axes in the same position they had when entered Hold.
0x0016001C	Process requires a not consistent instruction CNC tried to execute a command that is not allowed for the requesting process.
0x0016001D	Wrong FeedHold modality Unexpected FeedHold mode required.
0x0016001E	Unknown command class Internal error: Requested execution of a command belonging to an unknown class
0x0016001F	Unknown command Requested execution of an unknown command.
0x00160020	RoundOFF only for continuous movements RoundOFF command requested while system is not in continuous mode.

Error code (HEX)	Description
0x00160021	Pre-calculation elements finished. Command unaccepted. A continuous movement is requested while the pre-calculation queue is full.
0x00160022	Information not available Requested information about non-existing interpolators, processes or axes.
0x00160023	Spindle stopped The spindle is not moving and a threading or tapping operation is programmed. These operations require spindle rotation. Check S command programming and activation.
0x00160024	Spindle with acceleration ramps requested Tapping operation needs acceleration ramps (inversion time different from 0) to be configured for the spindle. In this way reversal can be performed at the bottom of the hole. Check spindle configuration using ODM tool.
0x00160025	Short tapping length Tapping length is not long enough to let the axes velocity be synchronised with spindle rotation speed.
0x00160026	Tapping axis acceleration wrong Tapping axis acceleration configured cannot reach tapping velocity respecting spindle reversal time.
0x00160027	Wrong spindle axis Programmed axes are not a spindle or it is a master / slave spindle.
0x00160028	Axis in use The axis is in use by another interpolator or process. Axis use request denied.
0x00160029	Axis in use during Hold movements The axis is in use by another interpolator or process moving it while in the Hold state. Axis use request denied.
0x0016002A	Axes with different clock Axes belonging to a movement, to an acquisition or to a virtualization have different interpolation clocks. They cannot be moved together. Check their configuration using ODM tool.
0x0016002B	Axis missing Requested axis is not configured. Check axis ID.
0x0016002C	Wrong axis ID Axis identifier is invalid, meaning that its value is out of range. Check axis ID.
0x0016002D	Request for Gantry axis (slave) use Axis ID requires the use of a Gantry axis (slave). Only master gantry axes can be used. Check axis ID and axis configuration using ODM tool.
0x0016002E	Axis not available Trying to move (during continuous) an axis that does not belong to interpolator or process.
0x0016002F	Duplicated axis Axis name requested more than once in a move or virtualization command. Check the instruction.

Error code (HEX)	Description
0x00160030	Axis in use by Master/Slave Axis requested belongs to a master/slave operation. It will be available once master/slave mode is complete.
0x00160031	Requested axis not available Requested axes are not associated with interpolator or do not belong to the requesting process.
0x00160032	Wrong Start condition for movement The condition for starting a movement is wrong. Check condition code and its associated parameters.
0x00160033	Wrong End condition for movement The condition for ending a movement is wrong. Check condition code and its associated parameters.
0x00160034	Wrong Break condition for movement The condition for interrupting a movement is wrong. Check condition code and its associated parameters.
0x00160035	Condition movement required Please specify at least a start/stop/break condition for the movement. Check the condition to be associated with the movement.
0x00160036	Wrong movement parameter The signal (bit type Boolean variable) associated with a start/stop/break condition for the movement is wrong or not compatible with the condition itself.
0x00160037	Wrong Move Until command In a Move Until command the requested execution mode is not allowed.
0x00160038	Wrong Start Continuous mode In the start continuous command the requested execution mode is not allowed.
0x00160039	Wrong End Continuous mode In the end continuous command the requested execution mode is not allowed.
0x0016003A	Wrong servo loop mode The requested servo mode is not allowed.
0x0016003B	Wrong Ramp mode Requested ramp mode is not allowed.
0x0016003C	Axis exceeds the positive limit Programmed position on the axis exceeds the positive limit, the command is not executed. Check command and current operational limits.
0x0016003D	Axis exceeds the negative limit Programmed position on the axis exceeds the negative limit, the command is not executed. Check command and current operational limits.
0x0016003E	Positive limit reached A movement reached positive limit. The running movement has been stopped.

Error code (HEX)	Description
0x0016003F	Negative limit reached A movement reached negative limit. The running movement has been stopped.
0x00160040	Axis on Positive Over travel Axis is on positive over travel. Movement command cannot be executed. Axis has to be manually moved within the operating range.
0x00160041	Axis on Negative Over travel Axis is on negative over travel. Movement command cannot be executed. Axis has to be manually moved within the operating range.
0x00160042	Reset required The instruction is active only if the error axis is reset.
0x00160043	Not spindle axis Request command can be executed only on a spindle axis. Check axis type.
0x00160044	Spindle unavailable Requested spindle does not belong to requesting interpolator or process.
0x00160045	Wrong gear value (spindle) The gear value used for the spindle axis exceeds allowed values. See ODM and available gears for spindle axis.
0x00160046	Axis missing Requested a movement with null feed, this value is not allowed. Check programmed value with F.
0x00160047	Wrong JRK value Programmed value for JRK (selection of characteristics for acceleration/deceleration with S ramps) is out of range.
0x00160048	Invalid variable used as condition to start a motion The variable used in start/stop/break condition of a movement is wrong or not compatible with the condition itself.
0x0016004B	Wrong feed mode Mode used to define work speed (Feed) is not available in the system. Work feed can use the following measurement units: length (mm or in) per minute, time, 1/time.
0x0016004C	Wrong SLL value Value for SLL variable (lowest spindle rotation speed in G96) cannot be negative. Check variable value.
0x0016004D	Wrong SSL value Value for SSL variable (highest spindle rotation speed in G96) cannot be negative. Check variable value.
0x0016004E	SSL value lower than SLL SSL variable (highest spindle rotation speed in G96) cannot be lower than SLL (lowest spindle rotation speed in G96).Check value of both variables.
0x0016004F	Function not available in tapping Tapping command requests a mode not allowed
0x00160050	Wrong origin number The origin requested for activation does not exist. The number is out of system range values.

Error code (HEX)	Description
0x00160051	Undefined origin Origin requested for activation is not defined.
0x00160052	Wrong Probing Safety Distance Value Safety distance for probing cannot be zero or negative. Check probing parameter value.
0x00160053	Wrong Probing Approach Distance Value Approach distance for probing cannot be zero or negative. Check probing parameter value.
0x00160054	Wrong Probing Feed Value Feed for probing cannot be zero or negative. Check probing parameter value.
0x00160055	Probing Executed in Approach Probing executed during approach phase, outside of the forecasted probing area.
0x00160056	Probing not Executed Probing not performed.
0x00160057	Probing in Execution Probing still executing.
0x00160058	Wrong probing status Probe displays an unexpected status. Contact the Customer Assistance.
0x00160059	Wrong probe size Probe ball diameter value cannot be negative. Check probe parameters.
0x0016005A	Unexpected ARM value Value for ARM variable is invalid. Check circles execution mode and possible corrections on the values programmed.
0x0016005B	Wrong helix move programming Value for helix step is not valid. Check helix programming and related K parameter.
0x0016005C	Wrong Rollover Axis programming Programmed position for rollover axis is invalid. Position has to be less than axis pitch. To perform multiple rotations on the axis, select incremental programming mode. Check rollover axis programming section.
0x0016005D	Movement inversion within a canned cycle Programmed positions for a canned cycle require a change in cycle direction. Check canned cycles programming section.
0x0016005E	Wrong lengths within a canned cycle Programmed positions for a canned cycle require lengths that are inconsistent with cycle characteristics. Check canned cycle programming section.
0x0016005F	Wrong Homing Direction The direction of the movement programmed for homing is inconsistent with axis configuration. Check, using ODM tool, homing configuration for axis.
0x00160060	Orienting spindle Gear cannot be changed while spindle is orienting.

Error code (HEX)	Description
0x00160066	Spindle without transducer Requested operation needs a spindle with transducer. Check function specification and, using ODM, spindle configuration.
0x00160067	Wrong thread length Length for thread operation is wrong and threading cannot be performed. Check threading cycle definition and programming.
0x00160068	Wrong thread pitch increment Pitch increment for thread operation is wrong and threading cannot be performed. Check threading cycle definition and programming.
0x00160069	Wrong hand wheel instruction Requested an invalid command with hand wheel.
0x0016006A	Wrong hand wheel scale value HandWheel motion required. The scale (distance to be covered for each hand wheel pulse) is wrong because it is negative. Negative values are not allowed.
0x0016006B	Wrong hand wheel pulses/revolutions number HandWheel manual motion required; the hand hweel is configured with a number of pulses/revolutions lower or equal to zero, that is an unaccepted value.
0x0016006C	Wrong hand wheel updating time HandWheel updating time cannot be lower or equal to 0.
0x0016006D	Wrong spindle sharing Sharing modality for spindle is wrong. Check available sharing modalities for spindle.
0x0016006E	Spindle acquisition denied Spindle is not configured as available for sharing, so sharing request is refused.
0x0016006F	Wrong status. Spindle sharing denied Request for sharing is accepted only when the process or the interpolator are not performing other operations (for example in continuous mode).
0x00160070	Rotation mode. Spindle sharing denied. Request for spindle sharing cannot be accepted since another interpolator or process owns t the spindle in exclusive mode.
0x00160071	Wrong axis sharing command Request axis sharing with wrong mode. Check available axis sharing.
0x00160072	Axis sharing release denied because of the PLC It is not possible to release the sharing condition of an axis when it is currently moved, in sharing mode, by another PLC interpolator. Switching to exclusive mode is available only when axis is idle.
0x00160073	Axis in use. Sharing mode denied It is not possible to release the sharing condition of an axis when it is currently moved, in sharing mode, by another process. Switching to exclusive mode is available only when axis is idle.
0x00160074	Wrong status. Sharing axis denied Sharing request is accepted only when requesting interpolator or process are not performing other actions (for example in continuous).

Error code (HEX)	Description
0x00160075	Axis shared. Release/acquisition denied. Request for acquiring/releasing an axis can be satisfied when axis is shared.
0x00160076	Axis not shared. Request denied. Request for sharing can be accepted only when axis owner makes the axis available for sharing.
0x00160077	Jerk move on negative profile Jerk for movement is negative and only positive values are accepted.
0x00160078	Negative acceleration motion requested Acceleration for movement is negative and only positive values are accepted.
0x00160079	Negative deceleration motion requested Deceleration for movement is negative and only positive values are accepted.
0x0016007A	Motion required with negative profile feed Feed for movement is negative and only positive values are accepted.
0x00160096	Move ended on positive software over travel Manual movement ended on positive limit.
0x00160097	Move ended on negative software over travel Manual movement ended on negative limit.
0x001600C9	Master/Slave crossed reference The axis to be set as a slave is already master of the requesting master. Check Master/Slave order and association.
0x001600CA	Not slave axis Requested a slave related operation on an axis that is not defined as a slave in a Master/Slave association.
0x001600CB	Operation not allowed on slave axis Requested operation cannot be performed since the slave is still linked to its master. To perform the action the master axis has to release the slave
0x001600CC	Release not allowed on slave axis Master cannot release slave since there are operations active on the slave.
0x001600CD	Wrong Master/Slave following mode Slave axis requested unavailable master following. Check available modes in Master/Slave section.
0x001600CE	Wrong Master/Slave activation or deactivation The requested Master/Slave activation or deactivation cannot be performed. Activation/deactivation mode is invalid. Check available modes in Master/Slave section.
0x001600CF	Master/Slave synchronisation on axis with different origins Requested a Master/Slave synchronisation depending on master position; Master/Slave origin should be the same for all slaves. Check Master/Slave programming
0x001600D0	Master/Slave synchronisation on axis with associated origin Requested a Master/Slave synchronisation depending on master position; Master/Slave origin can't be associated to the slaves. Check Master/Slave programming

Error code (HEX)	Description
0x001600D1	Slave axis use rejected Master / Slave association refused since slave axes already belongs to a dual axis association (UDA / SDA).
0x001600D2	Master axis use rejected Master axis does not belong to requesting process.
0x0016012C	Non-existent process Request for status of a non-existent process.
0x0016012D	LookAhead activation not allowed Request to start operation on a LookAhead already running.
0x0016012E	LookAhead instruction not allowed Tried to execute a command on a LookAhead that is not running
0x0016012F	CycleStart rejected Requested a Cycle start command when system is in a state that does not allow this command.
0x00160130	CycleStop rejected Requested a Cycle stop command when system is in a state that does not allow the movement stop (or the movement doesn't exist).
0x00160131	LookAhead instruction not consistent Requested a point-point (G29) movement during a continuous movement already running.
0x00160132	PLC process not existent An non-existent PLC process is required.
0x00161771	Axis homed End of homing operation with information about micro-marker distance (if the homing cycle uses the micro-marker search algorithm).

4. OPEN_LIBRARY: General purpose functions

AsciiToInt	Converts a character into its corresponding ASCII numerical value
IntToAscii	Converts a numerical value into an ASCII character
Decoder	Decodes some values in a word
Encoder	Encodes some bits in a word
InitData	Initialises a number of adjacent variables in an indexed manner
SearchData	Indexed search of a variable within a variables area
InitMemory	Initialises a memory zone
MoveData	Copies variables in indexed mode
MoveMemory	Copies data from a memory area to another
Delay	Introduces a delay
EnterCritical	Enters a critical zone with semaphore
ExitCritical	Exits a critical zone with semaphore
CriticalStatus	Checks if there are any WinPLUS tasks operating in critical zone
ReadInputsIntoWord	Reads a series of inputs into a word
WriteOutputsFromWord	Writes a set of outputs from a Word
SendMessage	Transmits a message on a specified queue
WarningMessage	Displays a warning message
SendSemaphore	Activates a WinPLUS task waiting at a semaphore
WaitOnSemaphore	Puts a WinPLUS task in wait at semaphore status
WaitOnSemaphoreStatus	Determines whether there are any WinPLUS tasks waiting at a semaphore
EmergencyStopClose	Closes the local E-STOP for the process communicated
EmergencyStopOpen	Opens the local E-STOP for the process communicated
FastInputRead	Reads fast inputs
FastOutputRead	Reads fast outputs
FastOutputWrite	Writes fast outputs
FileReadWrite	Supplies functions living access to formatted files
GetReleaseInfo	Provides the software version installed on the CNC system
GetSystemSerialNumber	Provides the Serial Number of the CNC system.
ME2_CommStatus	Reads the Mechatrolink I/II communication status
AckStrobeEmerg	Informs the system that an emergency has been acquired by the PLC
ClearStrobeEmerg	Resets the presence (for PLC only) of an emergency
GetStrobeEmerg	Determines the type of emergency present in the system
WinPlusEmergency	Generates a non-recoverable WinPLUS emergency
ReadFilterEmerg	Reads the data associated to an emergency warning
WaitOnMessage	Waits to receive a message
SetFilterEmerg	Requires whether to inform the PLC about system emergencies
WarningMessage	Displays a WARNING message
WarningMessageDisplay	Displays a series of warning messages by turns
TranslateError	Returns the message corresponding to a code error
TranslateErrorExt	Returns the message corresponding to a code error
GetBoardInfo	Supplies a description of the axis board resources
GetSystemInfo	Provides the CNC system information
OsWire_GetEmcyInfo	Reads data referred to the emergencies on I/O nodes connected on the OSWire Bus

ProbeDelete	Deletes a PLC diagnostic object
ProbeON	Sets the Probe signal at ON
ProbeCreate	Creates a PLC diagnostic object
ProbeOFF	Sets the Probe signal at OFF
GetAxisIDFromName	Determines the ID of an axis from its name
GetAxisNameFromID	Determines the name and the process of an axis
TBL_Lock	Requests exclusive access to a table
TBL_Unlock	Gives exclusive access to a table
TBL_ReadField	Reads a single field in a table record
TBL_WriteField	Allows to write a single field in a table record
TBL_ReadRecord	Reads a complete table record
TBL_WriteRecord	Writes a complete table record
TBL_SearchField	Allows to search a single field in a table
ConsoleOFF	Disconnects an operator console form a process
ConsoleON	Connects an operator console form a process
ConsoleReadInput	Reads the status of console inputs
ConsoleReadOutput	Reads the status of console outputs
ConsoleWriteOutput	Writes the status of console outputs
ConsoleSetup	Configures console selectors for a process in continuous mode
ConsoleSetupArray	Configures the operating modes of console selectors
Filter_Acc_Create	Creates a filter for the centripetal acceleration
Filter_Acc_Run	Inserts an acceleration filter
Filter_Bessel_Delete	Deletes a Bessel filter
Filter_Boost_Create	Creates a start or reversal filter
Filter_Boost_Run	Allows to run a start or reversal filter
Filter_LowPass1st_Delete	Deletes a first order low-pass filter
Filter_Smooth_Create	Creates a floating average filters
Filter_Smooth_Run	Allows to run a floating average filters
Filter_Acc_Delete	Deletes an acceleration filter
Filter_Bessel_Create	Creates a Bessel filter
Filter_Bessel_Run	Allows to run a Bessel filter
Filter_Boost_Delete	Deletes a start or reversal filter
Filter_LowPass1st_Create	Creates a first order low-pass filter
Filter_LowPass1st_Run	Allows to run a first order low-pass filter
Filter_Smooth_Delete	Deletes a floating average filters
LookUpT_Linear_Delete	Deletes a look-up table with linear interpolation
LookUpT_Spline3_Create	Calculates coefficients of a cubic spline
LookUpT_Spline3_Run	Allows to run an interpolation
LookUpT_Linear_Create	Calculates the linear interpolation coefficients
LookUpT_Linear_Run	Allows to run an interpolation
LookUpT_Spline3_Delete	Deletes a look-up table with linear interpolation
Pid_Delete	Deletes a PID controller
Pid_Create	Creates a PID controller
Pid_Run	Allows to run a PID controller
ECAT_CommStatus	Reads the EtherCAT communication status
ECAT_ReadDialogLog	Reads the EtherCAT communications software notifications
ECAT_WriteDeviceData	Allows to write information about a device connected through Sercos profile on an EtherCAT bus.
ECAT_ReadDeviceData	Reads information about a device connected through Sercos profile on an EtherCAT bus.

SOE_ReadParameter	Reads an IDN command on a device connected to an EtherCAT bus through Sercos profile
SOE_Command	Allows to run an IDN command on a device connected to an EtherCAT bus through Sercos profile
SOE_WriteParameter	Allows to write an IDN command on a device connected to an EtherCAT bus through Sercos profile
S3_Command	Allows to run an IDN command of a device connected on SERCOSIII bus
S3_ReadDeviceData	Reads data referring to a device connected on SERCOSIII bus
S3_ReadParameter	Reads an IDN parameter of a device connected on SERCOSIII bus
S3_WriteDeviceData	Allows to write data referring to a device connected on SERCOSIII bus
S3_CommsStatus	Reads SERCOSIII communication status
S3_ReadDialoLog	Reads the diagnostic in to the communication master
S3_WriteParameter	Allows to write a IDN parameter of a device connected on SERCOSIII bus

4.1 AsciiToInt

Function AsciiToInt converts a character into its corresponding ASCII numerical value.

Syntax:

```
ret_code := AsciiToInt (AsciiStr) ;
```

Input parameters:

AsciiStr (STRING[1]) Character to be converted

Execution mode:

Immediate

Use:

This function returns the numerical value (according to the ASCII table) of the input character. For example, by inputting character '0' to the function, we get value 0x30 hex (decimal 48) as output.

Return values:

The following table gives the possible values assumed by variable *ret_code*

ret_code	Description
n	Numerical value according to the ASCII table of the character input to the function

Example:

```
Ret : int;  
Ret := AsciiToInt ('0') ;
```

See also:

IntToAscii

4.2 IntToAscii

Function IntToAscii converts a numerical value into an ASCII character.

Syntax:

```
ret_code := IntToAscii (Value) ;
```

Input parameters:

Value (INT) Numerical value to be converted

Execution mode:

Immediate

Use

This function converts a numerical value into an ASCII character.



The value input as parameter is considered as being in the 0..255 range (ASCII characters range). Higher values are reduced to this range (upper part of value is truncated).

Return values:

<i>ret_code</i> (string)	Description
	1 character string containing the conversion of the <i>Value</i> value

Example:

```
Ret : string;
Ret := IntToAscii (48) ;
```

See also:

AsciiToInt

4.3 Decoder

The function Decoder decodes some values in a WORD.

Syntax:

```
ret_code := Decoder (DecodedVar, ValueToDecode, DecodingValues, ... ) ;
```

Input parameters:

ValueToDecode (WORD) Value to be decoded
DecodingValues (WORD....) Decoding values

Output parameters:

DecodedVar (WORD) Decoded value

Execution mode:

Immediate

Use:

When the value of parameter *ValueToDecode* is the same as the value of the parameter *DecodingValues*, the relative bit is set in *DecodedVar*, i.e., if it is the same as the first *DecodingValues*, bit 0 of *DecodedVar* is set, if it is the same as the second value, bit 1 is set, and so on. For this reason, no more than 16 *DecodingValues* may be present.

Return values

The following table gives the possible values assumed by variable *ret_code*

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0015000a	<i>ValueToDecode</i> not found between <i>DecodingValues</i>

Example:

```
Ret      : dword;
DecVar  : WORD;
Ret := Decoder (DecVar, 1, 10, 5, 1) ; (* DecVar will be= 0x0004 *)
```

4.4 Encoder

The function Encoder encodes some bits in a WORD

Syntax:

ret_code := Encoder (*EncodedVar*, *EncodingSignals*, ...) ;

Input parameters:

EncodingSignals (BOOL...) Signals to be encoded in the WORD

Output parameters:

EncodedVar (WORD) Decoding word

Execution mode:

immediate

Use:

This function encodes input signals in the variables *EncodingSignals* in position on the WORD *EncodedVar*. For this reason, no more than 16 *DecodingValues* may be present.

Return values

The following table gives the possible values assumed by variable *ret_code*

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0015000a	<i>EncodingSignals</i> all false

Example

```
Ret      : dword;
Encode   : WORD;
Ret := Encoder (Encode, true, false, false, true, true) ;
(* Encode = 0x19 *)
```

4.5 InitData

Function `InitData` initialises a number of adjacent variables in an indexed manner

Syntax:

```
ret_code := InitData (IndexIn, NumVar, Value, Data) ;
```

Input parameters:

<i>IndexIn</i> (INT)	Index of the initial variable
<i>NumVar</i> (INT)	Variables number to be initialised
<i>Value</i> (ANY_ELEMENTARY)	Initialisation value

Input/output parameters:

<i>Data</i> (ANY_ELEMENTARY)	Variable to be initialised
------------------------------	----------------------------

Possible overload:

- `InitData(INT, INT, BYTE, BYTE)`
- `InitData(INT, INT, WORD, WORD)`
- `InitData(INT, INT, INT, INT)`
- `InitData(INT, INT, REAL, REAL)`
- `InitData(INT, INT, DWORD, DWORD)`
- `InitData(INT, INT, DINT, DINT)`
- `InitData(INT, INT, LREAL, LREAL)`
- `InitData(INT, INT, STRING, STRING)`

Execution mode:

Immediate

Use:

This function may be used to initialise (in an indexed manner) a number of adjacent variables of the same type belonging to the same zone. Adjacent variables of the same type may be predefined WinPLUS variables, MWs, UDs ... and so on. It may also apply to WinPLUS local/global variables in array format. The initialisation process begins with the variable defined by *Data* and indexed by *IndexIn*, it is executed for *NumVar* elements. For example, to initialise 10 Words starting from MW17 we may recall the function and input variable MW0 as *Data* and value 17 as *IndexIn*, or MW17 as *Data* and value 0 as *IndexIn*.



The function does not check the validity of the indices of the variables used and therefore there is a risk of ending up writing outside the memory.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Ini : WORD := 0;
Arr : array [0..9] of WORD ;
(* Init from MW17 to MW26 with value 0 : two possible modes *)
Ret := InitData (17, 10, Ini, MW0) ;
Ret := InitData (0, 10, Ini, MW17) ;
(* Init array starting from the fifth element for 5 elements with value 0 *)
Ret := InitData (5, 5, Ini, array[0]) ;
Ret := InitData (0, 5, Ini, array[5]) ;
```

See also:

[InitMemory](#)

4.6 SearchData

SearchData function makes an indexed search of a variable within a variables zone.

Syntax:

```
ret_code := SearchData (IndexIn, NumVar, Value, Data, Position) ;
```

Input parameters:

<i>IndexIn</i> (INT)	Starting variable index
<i>NumVar</i> (INT)	variables to initialize
<i>Value</i> (ANY_ELEMENTARY)	Value for the initialization

Input/output parameters:

<i>Data</i> (ANY_ELEMENTARY)	Area where to search
<i>Position</i> (INT)	Position where the value was found

Possible overloads:

InitData(INT, INT, BYTE, BYTE, INT)
 InitData(INT, INT, WORD, WORD, INT)
 InitData(INT, INT, INT, INT, INT)
 InitData(INT, INT, REAL, REAL, INT)
 InitData(INT, INT, DWORD, DWORD, INT)
 InitData(INT, INT, DINT, DINT, INT)
 InitData(INT, INT, LREAL, LREAL, INT)
 InitData(INT, INT, STRING, STRING, INT)

Execution mode:

Immediate

Use:

Function can be used to search data within similar variables zone (indexed mode). Adjacent variables of the same type could be variables defined by WinPLUS as well as MW, UD ... and so on. WinPLUS local/global variables can be defined in array format.

The search starts from *Data* variable indexed by *IndexIn* and executed for *NumVar* elements. For example, to search a Word type data starting from MW17, it is possible to recall the function analysing the MW0 variable as *Data* and value 17 as *IndexIn*. Otherwise, MW0 variable can be analysed as *Data* and *IndexIn=0*.

Position will contain the position (starting from the search start position) to which the research value is referred. If value is not found, *Position* will be -1;



The function does not check the validity of the indices of the variables used and therefore there is a risk of ending up writing outside the memory.

Return value:

The following table lists all possible values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret      : dword;
Sea      : WORD := 33;
Arr     : array [0..9] of WORD ;
Found   : INT ;

(* Search from MW17 to MW26 for value 33 : two possible ways *)
Ret := searchData (17, 10, Sea, MW0, Found) ;
Ret := searchData (0, 10, Sea, MW17, Found) ;

(* Array search from fifth element for 5 elements having value 33 *)
Ret := searchData (5, 5, Sea, array[0], Found) ;
Ret := searchData (0, 5, Sea, array[5], Found) ;
```

4.7 InitMemory

Function `InitMemory` initialises a memory zone.

Syntax:

`ret_code := InitMemory (Offset, LenBytes, Value, Data) ;`

Input parameters:

<code>Offset</code> (INT)	Offset (in bytes) to be applied to the position
<code>LenBytes</code> (INT)	Bytes number to be initialised
<code>Value</code> (BYTE)	Initialisation value

Input/output parameters:

<code>Data</code> (ANY_ELEMENTARY)	Area to be initialised
------------------------------------	------------------------

Possible overload:

`InitMemory(INT, INT, BYTE, BYTE)`
`InitMemory(INT, INT, BYTE, WORD)`
`InitMemory(INT, INT, BYTE, INT)`
`InitMemory(INT, INT, BYTE, REAL)`
`InitMemory(INT, INT, BYTE, DWORD)`
`InitMemory(INT, INT, BYTE, DINT)`
`InitMemory(INT, INT, BYTE, LREAL)`
`InitMemory(INT, INT, BYTE, STRING)`

Execution mode:

Immediate

Use:

This function may be used to initialise a memory zone. The initialisation process begins with the variable defined by `Data` and indexed by `Offset` bytes, it is executed for `LenBytes` bytes. For example, to initialise 10 Words starting from MW17 we may recall the function and input variable MW0 as `Data` and value 34 as `Offset`, or MW17 as `Data` and value 0 as `Offset`.

In the case of STRING initialisation, the destination string will assume a length corresponding to the number of bytes initialised + starting offset (`LenBytes` + `Offset`).



The function does not check the validity of the indices of the variables used and therefore there is a risk of ending up writing outside the memory.

Return values:

<code>ret_code</code> (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Ini : BYTE := 0;
Arr : array [0..9] of LREAL ;
(* Init from MW17 to MW26 with value 0 : two possible modes *)
Ret := InitMemory (34, 20, Ini, MW0) ;
Ret := InitMemory (0, 20, Ini, MW17) ;
(* Init array fromthe fifth element for 2 elements with value 0  *)
Ret := InitMemory (5*8, 2*8, Ini, array[0]) ;
Ret := InitMemory (0, 2*8, Ini, array[5]) ;
```

See also:

InitData

4.8 MoveData

The function MoveData copies variables in indexed mode.

Syntax:

```
ret_code := MoveData (IndexIn, NumVar, IndexOut, DataIn, DataOut) ;
```

Input parameters:

<i>IndexIn</i> (INT)	Index to read variables from
<i>NumVar</i> (INT)	Number of variables to be transferred
<i>IndexOut</i> (INT)	Index to which the variables have to be transferred

Input/output parameters:

<i>DataIn</i> (ANY_ELEMENTARY)	Variable to be read from
<i>DataOut</i> (ANY_ELEMENTARY)	Variable to be written in

Possible overload:

MoveData(INT, INT, INT, BYTE, BYTE)
 MoveData(INT, INT, INT, WORD, WORD)
 MoveData(INT, INT, INT, INT, INT)
 MoveData(INT, INT, INT, REAL, REAL)
 MoveData(INT, INT, INT, DWORD, DWORD)
 MoveData(INT, INT, INT, DINT, DINT)
 MoveData(INT, INT, INT, LREAL, LREAL)
 MoveData(INT, INT, INT, STRING, STRING)

Execution mode:

Immediate

Use

This function transfers a number of variables in indexed mode from one memory area to another. The predefined variables WinPLUS as MW and UD variables can be considered as variables of the same type. It is also possible to use the local/global variables of WinPLUS.

The initialisation process begins with the variable defined by *DataIn* and indexed by *IndexIn*, it is executed for *NumVar* elements to reach the variable defined by *DataOut* indexed by *IndexOut*. For example, to copy 10 Words starting from MW17 to UW88 we may recall the function and input variable MW0 as *DataIn*, the value 17 as *IndexIn*, the variable UW0 as *DataOut* and the value 88 as *IndexOut*. Or it is possible to recall the variable MW17 as *DataIn*, the value 0 as *IndexIn*, the variable UW88 as *DataOut* and the value 0 as *IndexOut*.



This function does not perform any check on the validity of the variables used, and therefore there is a risk of writing outside the memory area.

Return value:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example

```
Ret : dword;
Ini : WORD := 0;
Arr : array [0..9] of WORD ;
(* Copy from MW17 to UW88 10 WORD : two possible modes *)
Ret := MoveData (17, 10, 88, MW0, UW0) ;
Ret := MoveData (0, 10, 0, MW17, UW88) ;
(* Copy array from the third element to the seventh for 2 elements *)
Ret := MoveData (3, 2, 7, array[0], array[0]) ;
Ret := MoveData (0, 2, 0, array[3], array[7]) ;
```

See also:

[MoveMemory](#)

4.9 MoveMemory

The function MoveMemory copies data from a memory area to another.

Syntax:

```
ret_code := MoveMemory (OffsetSrc, LenBytes, OffsetDest, Source, Dest) ;
```

Input parameters:

<i>OffsetSrc</i> (INT)	Offset to be applied to reading position
<i>LenBytes</i> (INT)	Number of bytes to be transferred
<i>OffsetDest</i> (INT)	Offset to be applied to writing position

Input/output parameters:

<i>Source</i> (ANY_ELEMENTARY)	Area to be read from
<i>Dest</i> (ANY_ELEMENTARY)	Area to transfer to

Possible overloads:

MoveMemory(INT, INT, INT, BYTE, BYTE)
MoveMemory(INT, INT, INT, BYTE, INT)
MoveMemory(INT, INT, INT, BYTE, WORD)
MoveMemory(INT, INT, INT, BYTE, DWORD)
MoveMemory(INT, INT, INT, BYTE, DINT)
MoveMemory(INT, INT, INT, BYTE, REAL)
MoveMemory(INT, INT, INT, BYTE, LREAL)
MoveMemory(INT, INT, INT, BYTE, STRING)
MoveMemory(INT, INT, INT, STRING, BYTE)

MoveMemory(INT, INT, INT, WORD, BYTE)
MoveMemory(INT, INT, INT, WORD, INT)
MoveMemory(INT, INT, INT, WORD, WORD)
MoveMemory(INT, INT, INT, WORD, DWORD)
MoveMemory(INT, INT, INT, WORD, DINT)
MoveMemory(INT, INT, INT, WORD, REAL)
MoveMemory(INT, INT, INT, WORD, LREAL)

MoveMemory(INT, INT, INT, INT, BYTE)
MoveMemory(INT, INT, INT, INT, INT)
MoveMemory(INT, INT, INT, INT, WORD)
MoveMemory(INT, INT, INT, INT, DWORD)
MoveMemory(INT, INT, INT, INT, DINT)
MoveMemory(INT, INT, INT, INT, REAL)
MoveMemory(INT, INT, INT, INT, LREAL)

MoveMemory(INT, INT, INT, DWORD, BYTE)
MoveMemory(INT, INT, INT, DWORD, INT)
MoveMemory(INT, INT, INT, DWORD, WORD)
MoveMemory(INT, INT, INT, DWORD, DWORD)
MoveMemory(INT, INT, INT, DWORD, DINT)

`MoveMemory(INT,INT,INT,DWORD,REAL)`
`MoveMemory(INT,INT,INT,DWORD,LREAL)`

`MoveMemory(INT,INT,INT,DINT,BYTE)`
`MoveMemory(INT,INT,INT,DINT,INT)`
`MoveMemory(INT,INT,INT,DINT,WORD)`
`MoveMemory(INT,INT,INT,DINT,DWORD)`
`MoveMemory(INT,INT,INT,DINT,DINT)`
`MoveMemory(INT,INT,INT,DINT,REAL)`
`MoveMemory(INT,INT,INT,DINT,LREAL)`

`MoveMemory(INT,INT,INT,REAL,BYTE)`
`MoveMemory(INT,INT,INT,REAL,INT)`
`MoveMemory(INT,INT,INT,REAL,WORD)`
`MoveMemory(INT,INT,INT,REAL,DWORD)`
`MoveMemory(INT,INT,INT,REAL,DINT)`
`MoveMemory(INT,INT,INT,REAL,REAL)`
`MoveMemory(INT,INT,INT,REAL,LREAL)`

`MoveMemory(INT,INT,INT,LREAL,BYTE)`
`MoveMemory(INT,INT,INT,LREAL,INT)`
`MoveMemory(INT,INT,INT,LREAL,WORD)`
`MoveMemory(INT,INT,INT,LREAL,DWORD)`
`MoveMemory(INT,INT,INT,LREAL,DINT)`
`MoveMemory(INT,INT,INT,LREAL,REAL)`
`MoveMemory(INT,INT,INT,LREAL,LREAL)`

Execution mode:

Immediate

Use:

Source and indexed by *OffsetSrc* bytes, it is executed for *LenBytes* bytes to reach the variable defined by *Dest* indexed by *OffsetDst* bytes. For example, to copy 10 Words starting from MW17 to UW88 we may recall the function and input variable MW0 as *Source*, the value 34 as *OffsetSrc*, the variable UW0 as *Dest* and the value 176 as *OffsetDst*. Or it is possible to recall the variable MW17 as *Source*, the value 0 as *OffsetSrc*, the variable UW88 as *Dest* and the value 0 as *OffsetDst*. If copying from BYTE to STRING, the destination string will be long as transferred number bytes + offset of destination in the string (*LenBytes*+ *OffsetDest*).



This function does not perform any check on the validity of the variables used, and therefore there is a risk of writing outside the memory area.

Return value:

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Ini : WORD := 0;
Arr : array [0..9] of LREAL ;
(* Copy from MW17 to UW88 10 WORD : two possible modes *)
Ret := MoveMemory (34, 20, 176, MW0, UW0) ;
Ret := MoveMemory (0, 20, 0, MW17, UW88) ;
(* Copy array from the third element to the seventh for 2 elements *)
Ret := MoveMemory (3*8, 2*8, 7*8, array[0], array[0]) ;
Ret := MoveMemory (0, 2*8, 0, array[3], array[7]) ;
```

See also:

MoveData

4.10 Delay

The function Delay introduces a delay.

Syntax:

```
ret_code := Delay (DelayTime) ;
```

Input parameters:

DelayTime (DWORD) Value of delay time with 10 ms base

Execution mode:

Immediate

Use:

Introduces a delay in the execution of PLC tasks for which it has been programmed. By programming a *DelayTime* of 1 introduces a 10ms delay.



If the delay programmed is 0, the task is suspended for an “infinite” length of time. It can be restarted only by executing a PLC warm/cold start.

Return value:

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Ret := Delay (10000) ;
```

4.11 EnterCritical

EnterCritical function enters a critical zone with semaphore.

Syntax:

```
ret_code := EnterCritical (Semaphore) ;
```

Input parameter:

Semaphore (INT) Semaphore number [0..63]

Execution mode:

Immediate / Wait

Use:

WinPLUS system has 64 semaphores (from 0 to 63) available to be used for synchronization operations within WinPLUS tasks. This function allows to enable a protection status (exclusive mode execution) of the operations executed after the call of the function. The protection is active until the function is called with *ExitCritical*. Protection operations are based on the use of semaphores; when entering a critical zone, semaphore is set to “red” so that all the other tasks are suspended; when exiting the critical zone, semaphore is set to “green” and one of the tasks can enter the safety area.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0X00150005	Wrong semaphore number

Example:

```
Ret : dword;
Ret := EnterCritical (10) ;
GW100 := 33; (* Protected assignments sequence *)
GW101 := 0;
Ret := ExitCritical (10) ;
```

See also:

[ExitCritical](#), [CriticalSection](#)

4.12 ExitCritical

ExitCritical function leaves a critical zone with semaphore.

Syntax:

```
ret_code := ExitCritical (Semaphore) ;
```

Input parameters:

Semaphore (INT) Semaphore number [0..63]

Execution mode:

Immediate

Use:

WinPLUS system has 64 semaphores(from 0 to 63) available to be used for synchronization operations within WinPLUS tasks. This function releases the use of a critical zone setting to “green” the semaphore identified by the *Semaphore* parameter. If more than one task retries to enter a critical zone, ExitCritical function restarts only one. If ExitCritical is used on a not-used semaphore (no task is waiting in “EnterCritical” on this semaphore) the function will set the semaphore in order to restart immediately the first task suspended on it.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0X00150005	Wrong semaphore number

Example:

```
Ret : dword;  
Ret := ExitCritical (10) ;
```

See also:

[EnterCritical](#), [CriticalStatus](#)

4.13 CriticalStatus

The CriticalStatus function verifies if there are any WinPLUS tasks working in a critical zone.

Syntax:

```
ret_code := CriticalStatus (Semaphore) ;
```

Input parameter:

Semaphore (INT) Semaphore number [0..63]

Execution mode:

Immediate

Use:

WinPLUS system has 64 semaphores (from 0 to 63) available to be used for synchronization operations within WinPLUS tasks. The function verifies if there are any WinPLUS tasks working in a critical zone, with *Semaphore* set on “red”.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	No task is waiting on the semaphore
0x00000001	There are some tasks waiting on the semaphore
0X00150005	Wrong semaphore number

Example:

```
Status : dword;
Status := CriticalStatus (10) ;
if Status = 0 then .... (*No task is waiting on semaphore 10 *)
```

See also:

[EnterCritical](#), [ExitCritical](#)

4.14 ReadInputsIntoWord

Function `ReadInputsIntoWord` reads a series of inputs in a WORD

Syntax:

```
ret_code := ReadInputsIntoWord (StartInputIndex, StartInputBit, NumberOfBits, ReadWord) ;
```

Input parameters:

`StartInputIndex` (INT) Index inputs zone word to read from

`StartInputBit` (INT) Bit to start reading from

`NumberOfBits` (INT) Number of inputs to be read [0..16]

Output parameters:

`ReadWord` (WORD) Variable in which the inputs read are returned

Execution mode:

Immediate

Use:

This function makes it possible to read a series of input bits from an I/O device. Reading starts from the `StartInputIndex` word of the input mapping zone; from this word, reading is executed starting from the `StartInputBit` signal and continuing for `NumberOfBits`. The function makes it possible to read inputs astride two different words. If parameter `NumberOfBits` exceeds the value of 16 (16 is the maximum number of inputs writable into the word), the system limits the reading to 16 Inputs. Parameter `StartInputBit` may be any positive value; values greater than 16 moves the reading on to inputs on words following the one defined by `StartInputIndex`.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150010	Reading exceeds inputs limits zone

Example:

```
Ret      : dword;
RdWrd   : WORD ;
Ret := ReadInputsIntoWord (10, 33, 8, RdWrd) ;
```

See also:

[WriteOutputsFromWord](#)

4.15 WriteOutputsFromWord

Function WriteOutputsFromWord writes a set of outputs from a WORD

Syntax:

```
ret_code := WriteOutputsFromWord (StartOutputIndex, StartOutputBit, NumberOfBits, WriteWord) ;
```

Input parameters:

<i>StartOutputIndex</i> (INT)	Word index in output writing zone
<i>StartOutputBit</i> (INT)	Initial write bit
<i>NumberOfBits</i> (INT)	Number of outputs to be written
<i>WriteWord</i> (WORD)	Word fromwhich Outputs are taken

Execution mode:

Immediate

Use:

This function makes it possible to write a set of output bits on the I/O devices. Writing occurs starting from the *StartOutputIndex* word in the Output mapping zone; fromthis word, writing takes place starting from the *StartOutputBit* signal and goes on for *NumberOfBits*. This function makes it possible to write outputs astride two different words. If parameter *NumberOfBits* exceeds the value of 16 (16 is the maximum number of Outputs writable with the word), the system limits the writing to 16 Outputs. Parameter *StartOutputBit* may be any positive value; values higher than 16, writing to Output is moved onto words following the one defined by *StartOutputIndex*.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150010	Reading exceeds outputs limits zone

Example:

```
Ret : dword;
Ret := WriteOutputsFromWord (10, 46, 8, 16#001C) ;
```

See also:

[ReadInputsFromWord](#)

4.16 SendMessage

The function SendMessage transmits a message on a specified queue.

Syntax:

```
ret_code := SendMessage (DestQueue, MsgSize, Messag) ;
```

Input parameters:

<i>DestQueue</i> (WORD)	Destination queue
<i>MsgSize</i> (INT)	Numbers of bytes to transmit
<i>Messag</i> (ARRAY of BYTE)	Numbers of MW to send

Execution mode:

Immediate

Use:

Through this function the WinPLUS task transmits a message on a queue of another WinPLUS task whose address is specified in *DestQueue*. This parameter may assume values of between 0x5101 and 0x51FB, where 0x5101 is the first PLC task, 0x5102 is the second,... 0x51FB is the two-hundred and fiftieth PLC task. The formula allowing to calculate the value to assign to *DestQueue* is 0x5100 + task index (value obtainable by means of function TSK_GetName). The message to be transmitted is uploaded in the *Messag* byte array. Different messages may be sent, one after the other, without waiting for the recipient task to read them.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150007	Parameters exceeding the size of areas MW, MD or A
0x00150008	Parameters exceeding the maximum size of the message (574 bytes)

Example

```
Ret : dword;
Msg : Array [0..10] of BYTE ;

(* Sends 2 INT (MI1, MI2) *)
COPY_INT_TO_BYTE_ARRAY (MI1, 0, Msg) ;
COPY_INT_TO_BYTE_ARRAY (MI2, 2, Msg) ;
Ret := SendMessage (16#5102, 4, Msg) ;
```

See also:

WaitMessage

4.17 WarningMessage

The function WarningMessage displays a message of Warning

Syntax:

ret_code := WarningMessage (*Message*, *OnHistory*) ;

Input parameters:

<i>Message</i> (ANY_ELEMENTARY)	Message identifier
<i>OnHistory</i> (INT)	Request to save message in system history file

Possible overloads:

WarningMessage (INT,INT)
WarningMessage (STRING,INT)

Execution mode:

Immediate

Use:

This function displays on screen a Warning message whose text is contained in the string passed in *Message* parameter (STRING) or contained in the predefined A variable with index passed in *Message* (INT). If parameter *OnHistory* is not 0, the message concerned shall be saved in the system history file, in the LOG message section.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0015000C	Wrong index variable A

Example:

```
Ret : dword;
A17 := 'Oil low' ;
Ret := WarningMessage (17, 0) ;      (*Send message to screen only *)
Ret := WarningMessage (A17, 0) ;     (*Send message to screen only *)
```

See also:

WarningMessageDisplay

4.18 SendSemaphore

Function SendSemaphore activates a WinPLUS task waiting at a semaphore.

Syntax:

```
ret_code := SendSemaphore (Semaphore) ;
```

Input parameters:

Semaphore (INT) Semaphore number [0..63]

Execution mode:

Immediate

Use:

The WinPLUS system has 64 semaphores (identified with a number from 0 to 63) which are used for the mutual synchronisation of WinPLUS tasks. This function allows WinPLUS tasks that are waiting to restart on the same semaphore -*Semaphore*- (several tasks may be waiting on the same semaphore). When you use SendSemaphore is used with an unused semaphore (no task is suspended in WaitOnSemaphore on this semaphore), the SendSemaphore function predisposes the semaphore so as to ensure that the first task that is suspended on it may start at once.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0X00150005	Number of semaphore wrong

Example:

```
Ret : dword;
Ret := SendSemaphore (10) ;
```

See also:

[WaitOnSemaphore](#), [WaitOnSemaphoreStatus](#)

4.19 WaitOnSemaphore

Function WaitOnSemaphore puts a WinPLUS task in wait at semaphore status.

Syntax:

```
ret_code := WaitOnSemaphore (Semaphore) ;
```

Input parameters:

Semaphore (INT) Semaphore number [0..63]

Execution mode:

Immediate / Wait

Use:

The WinPLUS system has 64 semaphores (identified with a number between 0 and 63) to be used for the mutual synchronisation of WinPLUS tasks. This function puts off the execution of a WinPLUS task until another task activates the SendSemaphore function on the same semaphore number. If function SendSemaphore has already been executed (i.e., the semaphore has already been activated), the task is not interrupted and its execution goes on.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0X00150005	Number of wrong semaphore

Example

```
Ret : dword;  
Ret := WaitOnSemaphore (10) ;
```

See also:

[SendSemaphore](#), [WaitOnSemaphoreStatus](#)

4.20 WaitOnSemaphoreStatus

Function `WaitOnSemaphoreStatus` determines whether there are any WinPLUS tasks waiting at a semaphore.

Syntax:

```
ret_code := WaitOnSemaphoreStatus (Semaphore) ;
```

Input parameters:

Semaphore (INT) Semaphore number [0..63]

Execution mode:

Immediate

Use:

The WinPLUS system has 64 semaphores (identified with a number between 0 and 63) to be used for the mutual synchronisation of WinPLUS tasks. The function determines whether there are any WinPLUS tasks waiting at a semaphore.

Return values:

ret_code (HEX)	Description
0x00000000	No task waiting on semaphore
0x00000001	Tasks waiting on the semaphore
0X00150005	Wrong semaphore number

Example:

```
Status : dword;  
Status := WaitOnSemaphoreStatus (10) ;  
if Status = 0 then .... (* No task suspended on semaphore 10 *)
```

See also:

[WaitOnSemaphore](#), [SendSemaphore](#)

4.21 EmergencyStopClose

Function EmergencyStopClose closes the local E-Stop for the process communicated.

Syntax:

```
ret_code := EmergencyStopClose (Process) ;
```

Input parameters:

Process (INT) Process number [0..24]

Execution mode:

Immediate

Use:

When a local E-stop is closed (when E-Stop contact is closed for one process and when global E-Stop is closed), the corresponding E-stop contact on the relay modulus closes. This closure is normally used to power up the drives and/or the machine magnetics. The global E-Stop will be energized by the system when the control is ready, but before PLC tasks start running. When control is ready, local E-Stops can now be energized by WinPLUS. If the function is activated with a process number outside the admissible range, the execution proceeds as if the function had been programmed with value 0 (i.e., all processes)

The general E-Stop will only be de-energized by the system in case of a non-recoverable error or emergency or in case of a hardware error. In this case, the E-Stop contact on the relay modulus opens. Subsequent *EmergencyStopClose* calls have no effect.

Return value

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Ret := EmergencyStopClose (0) ;
```

See also:

[EmergencyStopOpen](#)

4.22 EmergencyStopOpen

Function EmergencyStopOpen opens the local E-Stop contact for the process communicated

Syntax:

```
ret_code := EmergencyStopOpen (Process, Mode) ;
```

Input parameters:

Process (INT) Process number [0..24]

Mode (INT) Flag to open the E-Stop contact on the relay modulus

Execution mode:

Immediate

Use:

Through this function the local E-Stop contact is opened for the process specified. If the Mode parameter = 0, the E-Stop contact on the relay modulus is opened. If the function is activated with a process number outside the admissible range, the execution proceeds as if the function had been programmed with value 0 (i.e., all processes).

The E-Stop relay will only be de-energized by the system in case of a non-recoverable error or emergency or in case of a hardware error. In any case, the recoverable E-Stop contact opening emergency will be generated.

Return value:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;  
Ret := EmergencyStopOpen (0,0) ;
```

See also:

[EmergencyStopClose](#)

4.23 FastInputRead

Function FastInputRead is used to read the Fast Input.

Syntax:

```
ret_code := FastInputRead (FastInputStatus) ;
```

Output parameters:

FastInputStatus (WORD) Word where to read the Fast Inputs

Execution mode:

Immediate

Use:

This function reads the Fast Inputs as a function of the type of axis board present. The first bit of the word defines the first fast input, the second bit defines the second fast input, and so on according to the fast inputs present.



Not all boards support the fast inputs.

Return value:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret      : dword;
ReadInput : word ;
Ret := FastInputRead(ReadInput) ;
```

See also:

[FastOutputWrite](#), [FastOutputRead](#).

4.24 FastOutputRead

The function FastOutputRead is used to read the Fast Output.

Syntax:

```
ret_code := FastOutputRead (FastOutputStatus) ;
```

Output parameters:

FastOutputStatus (WORD) Word to read the Fast Output

Execution mode:

Immediate

Use:

This function reads the Fast Outputs as a function of the type of axis board present. The first bit of the word defines the first fast input, the second bit defines the second fast input, and so on according to the fast inputs present.



Not all boards support the fast outputs.

Return value:

<i>ret_code (HEX)</i>	Description
0x00000000	Function executed without errors

Example:

```
Ret      : dword;
ReadOutput: word ;
Ret := FastOutputRead (ReadOutput) ;
```

See also:

[FastOutputWrite](#), [FastInputRead](#).

4.25 FastOutputWrite

Function `FastOutputWrite` is used to write fast outputs.

Syntax:

```
ret_code := FastOutputWrite (FastOutStatus, OutMask) ;
```

Output parameters:

<i>FastOutStatus</i> (WORD)	Word containing the value of fast outputs
<i>OutMask</i> (WORD)	Mask describing the fast outputs to be written

Execution mode:

Immediate

Use:

This function writes the Fast Outputs as a function of the type of axis board present. The first bit of the *FastOutStatus* word defines the value of the first fast output to be written, the second bit defines the second fast output, and so on as a function of the number of fast outputs present. Word *OutMask* defines the fast outputs to be written, then, if the first bit is set to 1, the first fast output shall be written on the first bit, if the second bit is set to 1, the second fast output shall be written, and so on.



Not all boards support Fast Outputs.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Ret := FastOutputWrite (16#0002, 16#0003) ;
(*writes = 0 the first fast output and 1 the second *)
```

See also:

`FastOutputWrite`, `FastOutputRead`.

4.26 FileReadWrite

Function FileReadWrite supplies functions giving access to formatted files.

Syntax:

ret_code := FileReadWrite (*OpType*, *VarIndex*, *NumVar*, *FileName*, *VarType*, *Text*, *Offset*) ;

Input parameters:

<i>OpType</i> (INT)	Type of operation to be executed
<i>VarIndex</i> (WORD)	Predefined variable index
<i>NumVar</i> (WORD)	Number of variables to be used
<i>FileName</i> (STRING)	File pathname where you operate

Input/output parameters:

<i>VarType</i> (INT)	Type of predefined variable where you operate
<i>Text</i> (STRING)	Text to be written/read for variables not predefined
<i>Offset</i> (DINT)	Position to read/write on in a file

Execution mode:

Immediate

Use:

This function manages the read/write on file activities of "Predefined" WinPLUS variables (predefined variables include the GW, GD, MW, MD, UW, UD, ASCII, L, LS) as well as free format strings. The created/read file is sequential ASCII type and therefore can be accessed by any text editor. In the first line of a file (only in files containing predefined variables) a line is placed specifying the types of variables contained followed by a list of variables with the index format, value. Each line contains the data related to a single variable. Comments can be added to the file, according to the WinPLUS syntax for example: (* This is a comment *).

With files of this type, several operations can be performed: save, restore, request info, delete file.

The field *OpType* indicates the operation to be executed and takes on the following values:

OpType	Value	Description
op_STORE	1	Saves predefined variables in a file. If the file already exists, then a new file is created by making a backup copy of the previous one
op_RESTORE	2	Executes restore operations from a file of predefined variables
op_APPEND	3	Appends data to an existing file of predefined variables
op_INFO	4	Requests information regarding a file by returning the types of variable contained in it
op_CANCEL	5	Deletes a file
op_FREE_READ	6	Reads a text line from a file
op_FREE_APPEND	7	Appends a text line to an existing file
op_FREE_WRITE	8	Writes a text line to a file. If the file already exists, then a new file is created by making a backup copy of the previous one

Field **VarType** defines the types of variable to be written in a file (for write operations) or (for read or info operations) defines the types of variable contained in the file read; its value may be:

VarType	Value	Description	First line on file
vt_MW_VARS	1	Operation with predefined MW variable	MW VARIABLES
vt_MD_VARS	2	Operation with predefined MD variable	MD VARIABLES
vt_GW_VARS	3	Operation with predefined GW variable	GW VARIABLES
vt_GD_VARS	4	Operation with predefined GD variable	GD VARIABLES
vt_A_VARS	5	Operation with predefined A variable	ASCII VARIABLES
vt_LOG_VARS	6	Operation with variable of type string	MESSAGES *
vt_UW_VARS	7	Operation with predefined UW variable	UW VARIABLES
vt_UD_VARS	8	Operation with predefined UD variable	UD VARIABLES
vt_L_VARS	9	Operation with predefined L variable	L VARIABLES
vt_LS_VARS	10	Operation with predefined LS variable	LSVARIABLES

* If the operations are executed through functions *op_FREE_xxx* the first line is omitted.

The parameters described herein are always necessary for the execution of this function; the other parameters are not always required, and it is necessary to look them up in the following table to determine whether they are required or not, and as a function of which *OpType* they have to be specified.

Variables	Operation									
	Writing		Reading		Append		Info		Delete	
	Var.	Text	Var.	Text	Var.	Text	Va. r.	Text	Var.	Text
<i>VarIndex</i>	X	-	X	-	X	-	-	-	-	-
<i>NumVar</i>	X	-	-	-	X	-	-	-	-	-
<i>FileName</i>	X	X	X	X	X	X	X	X	X	X
<i>VarType</i>	X	X	X	-	X	X	X	X	-	-
<i>Text</i>	-	X	-	X	-	X	-	-	-	-
<i>Offset</i>	-	-	-	X	-	-	-	-	-	-

Legend: X = Mandatory; - = Not used.

Write operation (*OpType* = *op_STORE*)

This operation makes it possible to write variables or text in the file defined with the "FileName" parameter. The path can also be defined in the file name. For example: "E:\DIRLOG\file.txt". If the file does not exist, it is created, otherwise, before writing, the system renames the existing file and appends the .back extension to it.

When dealing with predefined WinPLUS variables, the parameters used by the function are:

1. **VarIndex** (Index of the first variable to be written);
2. **NumVar** (Number of variables to be written, starting from VarIndex);
3. **FileName** (File name or complete path name);
4. **VarType** (Type of variable to be written).

A header is written in the first line of the file created, indices and the relative values of the variables are written from the second line on. The format is as follows:

Index	Value or string	Hexadecimal value (an option, only for MW, UW and GW)
-------	-----------------	---

For instance, if the writing of 4 variables (NumVar = 4) MW (VarType = vt_MW_VARS) starting from index 10 (VarIndex = 10) is requested, the file (FileName = \SSD\MyFile.txt) will contain the following data:

MW VARIABLES

10	233	E9
11	10	A
12	255	FF
13	25374	631E

If a hexadecimal value is modified by a text editor and the decimal value does not change, during the reset the decimal value used.

If you want to write text in a file, the parameters to be supplied as inputs to the function will be:

1. **FileName** (File name or complete path name);
2. **VarType** (=vt_LOG_VARS);
3. **Text** (Text to be written).

A header is written in the first line of the file created, the text supplied to the function through the Text parameter is written in the second line. To be able to write more lines in the same file, see the Append operation.

Read or restore operation (OpType = op_RESTORE)

This operation makes it possible to restore the predefined WinPLUS variables with the values saved in the file, or to read text lines. The file must exist, otherwise the function returns an error. The first line (file header) is used solely to assess the congruence between the type of variable requested and the type present in the file.

For predefined WinPLUS variables, the parameters used by this function are:

1. **VarIndex** (Index of the first variable to be read);
2. **FileName** (File name or complete path name);
3. **VarType** (Type of variable to be read)

Parameter VarIndex makes it possible to identify the variable index to be used in restoring the values:

VarIndex	Restore index
0	Same as indicated in the file
>0	Determined as: (file index) + (VarIndex). The new code must be in the appropriate range as a function of type of variable to be restored.

Example:

Let us consider the example made for the MW variables in the write operation.

If VarIndex = 0, the values will be restored in the MW10, MW11, MW12 and MW13 variables.

If VarIndex = 5, the values will be restored in the MW15, MW16, MW17 and MW18 variables.

If we want to read text from a file, the parameters to be supplied as inputs to the function are:

1. **FileName** (File name or complete path name);
2. **VarType** (vt_LOG_VARS);

In this case, the file is read one line at a time. The line read (max 128 byte) is supplied as an input to the function with the "Text" parameter. The offset (expressed in bytes) of the following line in the file is also provided as an output from the function, this value is supplied as input to the next function call to be able to read the next text line.

Append operation (OpType = op_APPEND)

This operation writes a predefined variable or a text at the end of the file. The file must exist and be consistent with the type of variable to be written.

For predefined WinPLUS variables, the parameters used by this function are:

1. **VarIndex** (Index of the first variable to be written);
2. **NumVar** (Number of variables to be written from VarIndex);
3. **FileName** (File name or complete path name);
4. **VarType** (Type of variable to be written).

Starting from the end of the file, the indices and the relative values of the variables are written. The format is as follows:

Index	Value or string	Hexadecimal value (optional only for MW, UW and GW)
-------	-----------------	---

If we want to add text to a file, the parameters to be supplied as inputs to the function are:

1. **FileName** (File name or complete path name);
2. **VarType** (vt_LOG_VARS);
3. **Text** (Text to be written).

Starting from the end of the file, the input text is added.

Info operation (OpType = op_INFO)

This operation determines the types of variable contained in the file. The information is as supplied as an output in the VarType parameter.

The parameters to be supplied as inputs to the function are:

1. **FileName** (File name or complete path name);

Delete operation (OpType = op_DELETE)

This operation deletes an existing file. The parameters to be supplied as inputs to the function are:

1. **FileName** (File name or complete path name);

Reading operation (OpType = op_FREE_READ)

This operation reads text lines contained in a file. If present, the first line (header) is read as a regular text line.

The parameters to be supplied as inputs to the function are:

1. **FileName** (File name or complete path name);
2. **VarType** (vt_LOG_VARS);

In this case, the file is read one line at a time. The line read (max 128 byte) is supplied as an input to the function with the "Text" parameter. The offset (expressed in bytes) of the following line in the file is also provided as an output from the function, this value is supplied as input to the next function call to be able to read the next text line.

Writing operation (OpType = op_FREE_APPEND)

This operation adds text lines to the file specified in parameter *FileName*. The name must also indicate the path. Example: "E:\DIRLOG\file.txt". The file must exist.

The parameters to be supplied as inputs to the function are:

1. **FileName** (File name or complete path name);
2. **VarType** (vt_LOG_VARS);
3. **Text** (Text to write).

Writing operation (OpType = op_FREE_WRITE)

This operation makes it possible to write variables or text in the file defined with the "FileName" parameter. The path can also be defined in the file name. For example: "E:\DIRLOG\file.txt".

If the file does not exist, it is created, otherwise, before writing, the system renames the existing file and appends the .back extension to it. The first line (header) of the file is NOT created.

The parameters to be supplied as inputs to the function are:

1. **FileName** (File name or complete path name);
2. **VarType** (vt_LOG_VARS);
3. **Text** (Text to write).

Append operation writes several text lines in the same file.



Do not perform write/restore operations from/to MD, UD and GD variables containing the array dimensions possibly defined in them. Improper use will cause a system error or logic malfunction.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Error while opening file
0x00150015	Parameter VarType wrong
0x00150016	Parameter OpType wrong
0x00150017	Predefined variable writing out of range (Warning)
0x00150018	Write variable incongruent with the variables present in the file
0x00150019	Error on first line file (header)
0x0015001A	File does not contain predefined variables
0x0015001B	Predefined variable reading out of range
0x0015001C	Error reading file
0x0015001D	Reached end of file while reading

Example:

```

Ret      : dword;
VarType  : INT ;
Text     : STRING ;
Offset   : DINT ;

(*Writes 100 MW starting from 33 *)
VarType := vt_MW_VARS ;
Offset  := 0 ;
Text    := '' ;
ret := FileReadWrite (op_STORE, 33, 100, '\SSD\MyFile', VarType, Text,
Offset) ;

```

4.27 GetReleaseInfo

GetReleaseInfo function provides the software release installed on the CNC system.

Syntax

```
ret_code := GetReleaseInfo (Major, Minor, Fixup, RelStr) ;
```

Output parameters

<i>Major</i> (INT)	Installed release major version
<i>Minor</i> (INT)	Installed release minor version
<i>Fixup</i> (INT)	Fixup level installed
<i>RelStr</i> (STRING)	Complete string identifying the installed release

Execution mode

Immediate

Use

Provides data identifying the installed release within the variable.

Return values

The following table list all possible that values *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example

```
Ret : dword;
Major : INT;
Minor : INT ;
Fixup : INT ;
RelStr : STRING ;
Ret := GetReleaseInfo (Major, Minor, Fixup, RelStr) ;
```

4.28 GetSystemSerialNumber

The function GetSystemSerialNumber provides the Serial Number of the CNC system.

Syntax

```
ret_code := GetSystemSerialNumber (SerialNo) ;
```

Output parameter

SerialNo (STRING) String complete identifying the CNC Serial Number

Execution mode

Immediate

Use

Within the variable, it provides the data identifying the serial number of the CNC.

Return values

The following table list all possible values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example

```
Ret : dword;
Serial : STRING ;
Ret := GetSystemSerialNumber (Serial) ;
```

4.29 ME2_CommStatus

ME2_CommStatus function reads Mechatrolink I/II communication status.

Syntax

```
ret_code := ME2_CommStatus (BoardNumber, CommStatus);
```

Input parameters

BoardNumber (INT) Board number

Output parameters

CommStatus (STRUCT) This structure contains information about the Mechatrolink I/I communication status

Use

The function reads some data on Mechatrolink I/II field bus communication status. Information are grouped in the following *ME2_CommStatusStruct* structure:

Field	Type	Description
Ests	WORD	JL-080 component word status
Reserved1	WORD	Reserved
Reserved2	WORD	Reserved
Reserved3	WORD	Reserved

The word **Ests** contains the following error flags:

Bit	Mask (Hex)	Name	Description
0	0x0001	ME2_CRCERR	CRC error
1	0x0002	ME2_ABTERR	Abort error
2	0x0004	ME2_PARERR	Invalid communication parameter
3	0x0008	ME2_TMCYCOVR	Transmission cycle time overrun
4	0x0010	ME2_ALGERR	Alignment error
5	0x0020	ME2_SPTERR	Short packet errors
10	0x0400	ME2_TGLERR	Buffer exchange error
11	0x0800	ME2_TURERR	Transmission in error
12	0x1000	ME2_RFOERR	Receipt FIFO queue in error
14	0x4000	ME2_EWDTOVR	External watchdog released
15	0x8000	ME2_WDTOVR	Watchdog released

When all error flags are 0 the communication is correct.

Return value

Value (HEX)	Description
0	Function executed without errors

Example

VAR

```
Ret_code: DWORD;
Status : ME2_CommStatusStruct;
END_VAR
Ret_code := ME2_CommStatus (0, Status);
```

4.30 AckStrobeEmerg

Function AckStrobeEmerg informs the system that an emergency has been acquired by the PLC.

Syntax:

```
ret_code := AckStrobeEmerg (Event, AckNack) ;
```

Input parameters:

Event (DINT) Type of emergency whose acquisition is notified.
AckNack (BOOL) Acknowledge given/not given to system for event notified.

Execution modality:

Immediate

Use:

If the management of Emergency filters has been enabled (see SetFilterEmerg), any emergency generated in the system is communicated to the PLC. The moment the PLC has acquired the emergency (see GetStrobeEnerg), the PLC must inform the system of the acquisition by giving out an acknowledge. Upon receiving this acknowledge the system will be able to notify any subsequent emergency alerts.



The system will freeze if the PLC does not reply to an emergency alert with the AckStrobeEmerg function. Needless to say, this is so, provided that the PLC has activated the filters management option on emergency events.

Event defines the type of emergency responding to and its value will be:

Event	Value	Description
RECOVERABLE_EMERGENCY	0x00000001	Acknowledge for recoverable emergency
UNRECOVERABLE_EMERGENCY	0x00000002	Acknowledge for not recoverable emergency
ERROR_BROADCAST	0x00000004	Acknowledge for error notification
WARNING_BROADCAST	0x00000008	Acknowledge for warning notification

The **AckNack** parameter defines the response mode and its value will be:

AckNack	Value	Description
ACK	true	Emergency accepted by PLC
NACK	False	Emergency rejected by PLC

Regardless of whether the PLC acknowledges/does not acknowledge an emergency, the system does not take it into account and carries on its activities regularly. This parameter may be supported in future releases.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Event parameter wrong

Example:

```
Ret : dword;  
Ret := AckStrobeEmerg (RECOVERABLE_EMERGENCY,ACK) ;
```

See also:

[SetFilterEmerg](#), [GetStrobeEmerg](#), [ClearStrobeEmerg](#), [ReadFilterEmerg](#)

4.31 ClearStrobeEmerg

Function ClearStrobeEmerg resets the presence (for PLC only) of an emergency.

Syntax:

```
ret_code := ClearStrobeEmerg (Event) ;
```

Input parameters:

Event (DINT) Type of emergency whose presence has to be cleared.

Execution mode:

Immediate

Use:

This function clears the presence of an emergency communicated to the PLC. The moment an emergency is generated, the system regards it as present and available for the PLC until the PLC acknowledges its acquisition with function AckStrobeEmerg. At this point, the system is ready to generate further emergencies.

To stop GetStrobeEmerg function communicating any emergency present for the PLC and to keep the emergency management frozen, use the ClearStrobeEmerg function

The *Event* parameter defines the type of emergency to be cleared, and it may be:

Event	Value	Description
RECOVERABLE_EMERGENCY	0x00000001	Acknowledge for recoverable emergency
UNRECOVERABLE_EMERGENCY	0x00000002	Acknowledge for not recoverable emergency
ERROR_BROADCAST	0x00000004	Acknowledge for error notification
WARNING_BROADCAST	0x00000008	Acknowledge for warning notification



The system will freeze if the PLC does not reply to an emergency alert with the AckStrobeEmerg function. Assuming that the PLC has activated the filter management option for emergency events.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Event parameter wrong

Example

```
Ret : dword;
Ret := ClearStrobeEmerg (RECOVERABLE_EMERGENCY) ;
```

See also:

[SetFilterEmerg](#), [GetStrobeEmerg](#), [AckStrobeEmerg](#), [ReadFilterEmerg](#)

4.32 GetStrobeEmerg

Function GetStrobeEmerg makes it possible to determine the type of emergency present in the system.

Syntax:

```
ret_code := GetStrobeEmerg (Event) ;
```

Output parameters:

<i>Event</i> (DINT)	Emergency type to be cancelled
---------------------	--------------------------------

Execution mode:

Immediate

Use:

If the filter on emergency management function has been enabled (see SetFilterEmerg), each emergency generated in the system is communicated to the PLC. This function can determine the type of emergency present in the system.

The type of emergency present in the system will be communicated in the *Event* variable which is input as parameter to the function. This variable will contain the following values:

Event	Value	Description
RECOVERABLE_EMERGENCY	0x00000001	Acknowledge for recoverable emergency
UNRECOVERABLE_EMERGENCY	0x00000002	Acknowledge for not recoverable emergency
ERROR_BROADCAST	0x00000004	Acknowledge for error notification
WARNING_BROADCAST	0x00000008	Acknowledge for warning notification

Return value:

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret    : dword;
Event  : dword;
Ret := GetStrobeEmerg (Event) ;
```

See also:

[SetFilterEmerg](#), [AckStrobeEmerg](#), [ClearStrobeEmerg](#), [ReadFilterEmerg](#)

4.33 WinPlusEmergency

Function WinPlusEmergency generates a NON recoverable WinPLUS emergency.

Syntax:

```
ret_code := WinPlusEmergency (TaskNum, EmergCode);
```

Input parameters:

TaskNum (INT) Number of WinPLUS task for which the emergency is generated
EmergCode (DINT) Emergency number

Execution mode:

Immediate

Use:

This function generates a NON recoverable emergency of the type “WinPLUS Exception”. This emergency will be associated with WinPLUS task number *TaskNum* (task name obtainable by means of function TSK_GetName) with value *EmergCode*. It is good practice not to use an *EmergCode* with a value of between 32 and 50, since these are used directly by WinPLUS.



As the emergency is NON recoverable, a reboot is needed to restart the system.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Ret := WinPlusEmergency ( 0, 100 ); (* Generates WinPLUS 100 emergency *)
```

4.34 ReadFilterEmerg

Function ReadFilterEmerg reads the data associated with an emergency warning.

Syntax:

```
ret_code := ReadFilterEmerg (EmergData) ;
```

Output parameters:

EmergData (ANY) Structure in which the data regarding an emergency warning are to be read

Possible overloads:

- ReadFilterEmerg (RecEmerg_FilterStruct)
- ReadFilterEmerg (UnRecEmerg_FilterStruct)
- ReadFilterEmerg (Error_FilterStruct)
- ReadFilterEmerg (Warning_FilterStruct)

Execution mode:

Immediate

Use:

When the system communicates an emergency (whose communication was requested through function SetFilterEmerg), the data associated with it can be read. The data are read on the basis of the structure input, i.e., if the input is a structure type RecEmerg_FilterStruct, the data regarding a recoverable emergency are read, whereas, if the structure input is type UnRecEmerg_FilterStruct, the data regarding a NON recoverable emergency are read. The type of emergency notified and present in the system may be determined by means of function GetStrobeEmerg.



In order to be warned of an emergency and read the relative data the management thereof must have been enabled through function SetFilterEmerg. If the data are read without having enabled the management of the emergency, the values returned may be insignificant.

If function GetStrobeEmerg returns the presence of a RECOVERABLE_EMERGENCY event, the emergency will have to be read in a type RecEmerg_FilterStruct structure organised as follows:

Field	Type	Description
EmCode	INT	Code of emergency communicated (see following table)
Process	WORD	Process that has generated the emergency. Process = 0 stands for PLC-generated emergency
ProInErr	WORD	Emergency recipient process. With process = -1 the emergency is sent to all processes, with process = 0 the emergencies go to the PLC
ExtParam1...ExtParam4	INT	Correlated to type of emergency, see description in the application manual.
ExtParam5..ExtParam6	LREAL	Correlated to the error type, see description in the application manual.
Extra	Array BYTE	Extra information about the emergency. The information can be sent to the function TranslateErrorExt as to get an emergency description string.

If function GetStrobeEmerg returns the presence of a UNRECOVERABLE_EMERGENCY event, the emergency will have to be read in a type UnRecEmerg_FilterStruct structure organised as the previous one.

In the case of a recoverable emergency, the value of the EmCode field may assume the following values:

EmCode	Value	Description
e_SERVO_ER	101	Servo error
e_SKEW_ER	102	Skew error
e_AXIS_TOLERANCE_ER	103	Error on tolerance entry
e_AXIS_MARKER_ER	104	Marker too distant from micro
e_AXIS_INITIALIZ_ER	105	Axes initialization error
e_AXIS_CONFIG_ER	106	Axis configuration error
e_AXIS_COMMAND_ER	107	Axis command error
e_LOPMAN_IN_ER	120	Software overtravel reached in manual mode
e_LOPMAN_OUT_ER	121	Manual return from software over travels
e_OVERTR_IN_ER	122	Over travel input
e_OVERTR_OUT_ER	123	Output Over travel axes
e_DRV_INIT_ER	130	Drive initialization error
e_DRV_COMND_ER	131	Error on command request to drive
e_DRV_ALARM_ER	132	Drive alarm
e_DRV_COMMIC_ER	133	Communication error with drive
e_SENSOR_COMMERR	140	Communication error with sensor
e_SENSOR_HTTERR	141	Hardware Tip Touch error
e_SENSOR_SWNERR	143	Distance error under normal Timeout threshold
e_SENSOR_TIPERR	144	Tip functioning error
e_INTP_AX_ER	180	Emergency for axes motion interpolator
e_INTP_PG_ER	181	Interpolator emergency
e_PLCESTOP_ER	190	PLC E-Stop
e_SYST_ESTOP_ER	191	System E-stop

In the case of a recoverable emergency, the value of the EmCode field may assume the following values:

EmCode	Value	Description
e_ENCODER_FAULT_AN	1	Fault on Encoder
e_ANOMALY_AXES_AN	2	Axes anomaly
e_CAN_IOCOMM_AN	20	CAN I/O communication error
e_OSW_IOWATCHDOG_AN	30	OSWire I/O watchdog
e_OSW_IOCOMM_AN	31	OSWire I/O communication error
e_OSW_DRVFAULT_AN	32	OSWire drive fault
e_WATCHDOG_AN	80	System Watch Dog
e_POWER_FAIL_AN	81	Power fail on axes board
e_EXCP_WINPLUS_AN	82	Exception WinPLUS
e_WATCHDOG_BOARD_AN	83	Watch Dog from axis board
e_SHUTDOWN_AN	90	Shutdown
e_SYSANOM_AN	98	System anomaly
e_SYSEXCP_AN	99	System exception

If function GetStrobeEmerg returns the presence of an ERROR_BROADCAST event, the error has to be read within an Error_FilterStruct structure type as follow:

Field	Type	Description
Ambient	WORD	Ambient causing the error.
Code	WORD	Error code. See error list in the application manual.
Process	WORD	Process to which the error is referred. If process is 0, error refers to PLC.
CommSubcom	WORD	Instruction/sub-instruction causing the error. This field is mainly for internal system use.
ExtParam1...ExtParam4	INT	Correlated to the error type, see description in application manual.
ExtParam5..ExtParam6	LREAL	Correlated to the error type, see description in application manual.
Extra	Array of BYTE	Array containing all the information about the error. Information can be sent to the function TranslateErrorExt to get an error description string.
Flags	WORD	Extra flags associated with to the error

The field Flags can have several meanings:

Flags	Value	Description
e_SYSHIST	0x0001	Error reported in the System History file
e_WARNINGERR	0x0002	Warning type error

If function GetStrobeEmerg returns the presence of a WARNING_BROADCAST event, the error has to be read within a Warning_FilterStruct structure type as follow:

Field	Type	Description
Code	WORD	Warning code. (See following table)
ExtParam1...ExtParam4	INT	Correlated to the error type, see description in application manual.
ExtParam5..ExtParam6	LREAL	Correlated to the error type, see description in application manual.
Extra	Array of BYTE	Array containing all the information about the warning. Information can be sent to the function TranslateErrorExt to get a warning description string.

Code filed can have the following value:

Code	Value	Description
e_CANOPEN_HILSCHER_WARN	201	CANopen Hilscher warning
e_SERCOS_WARN	202	SERCOS warning
e_CANOPEN_PASSIVE_WARN	204	CANopen passive warning
e_PROFIBUS_MASTER_WARN	206	Profibus master warning
e_PROFIBUS_SLAVE_WARN	207	Profibus slave warning

Return value:

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```

Ret      : dword;
NRecov  : UnRecEmerg_FilterStruct;
Recov   : RecEmerg_FilterStruct;
ErrorB  : Error_FilterStruct;
ErrorW  : Warning_FilterStruct;
Event    : dint ;

Ret      := GetStrobeEmerg(Event);
CASE Event OF
  UNRECOVERABLE_EMERGENCY :
    Ret := ReadFilterEmerg (NRecov);
  RECOVERABLE_EMERGENCY   :
    Ret := ReadFilterEmerg (Recov);
  ERROR_BROADCAST        :
    Ret := ReadFilterEmerg (ErrorB);
  WARNING_BROADCAST      :
    Ret := ReadFilterEmerg (ErrorW);
END_CASE;

```

See also:

[SetFilterEmerg](#), [GetStrobeEmerg](#), [AckStrobeEmerg](#), [ClearStrobeEmerg](#), [TranslateError](#), [TranslateErrorExt](#)

4.35 WaitOnMessage

WaitOnMessage function waits for a message.

Syntax

```
ret_code := WaitOnMessage (Timeout, Messag, MsgLen, MsgFrom) ;
```

Input parameters

<i>Timeout</i> (DINT)	Response timeout (in 0,1 ms) if =0 never ending wait
<i>Messag</i> (ARRAY of BYTE)	Message reading area

Output parameters

<i>MsgLen</i> (INT)	Number of characters received
<i>MsgFrom</i> (WORD)	Id of the task sending the message

Execution mode

Immediate / Wait

Use

With this function WinPLUS task puts in hold a message from another PLC task. Once message is arrived, it can define the recipient as specified in *MsgFrom*. Parameter values have a range from 0x5101 to 0x51FB, where 0x5101 is the first PLC task, 0x5102 is the second PLC task,... 0x51FB is the two-hundred fiftieth PLC task. The formula calculating the queue number is 0x5100 + task index (values to be determined using the TSK_GetName function). Message is made by BYTES read within the array *Messag*, the number of characters will be in the *MsgLen* variable. If the message is already on the WinPLUS task queue, it is read immediately and the task keeps on running; if the message is missing, task dwells until the response or until the *Timeout* is active.

Return values

The following table lists all possible values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function Executed without errors
0x00150009	Indices/data exceed the MW, MD or Aareas dimensions
0x00150006	During system reboot WaitOnMessage is denied

Example

```
Ret    : dword;
From   : WORD ;
Len    : INT ;
Msg    : Array [0..10] of BYTE ;

(* Reads 2 INTERI in (MI1, MI2) *)
Ret := WaitMessage (0, Msg, Len, From) ;
MI1 := COPY_INT_FROM_BYTE_ARRAY (0, Msg) ;
MI2 := COPY_INT_FROM_BYTE_ARRAY (2, Msg) ;
```

See also:

SendMessage

4.36 SetFilterEmerg

Function SetFilterEmerg determines whether or not the PLC is informed of emergencies occurring in the system.

Syntax

```
ret_code := SetFilterEmerg (Event, Enable) ;
```

Input parameters

Event (DINT)	Types of emergency of which the PLC must be informed
Enable(BOOL)	Enable/Disable the functionality
PlcFiltEnab(BOOL)	Filter enabling for PLC request as well

Execution mode:

Immediate

Use:

Enables/disables the use of emergency filters. If a filter is enabled, the PLC is informed of any emergency that comes into being and for each the PLC must inform the system that the emergency has been acquired (see AckStrobeEmerg). If a filter is disabled, the system does not inform the PLC of the emergencies.



Once the management of the filter in emergency is enabled, the system will freeze if the PLC does not reply to an emergency alert with the AckStrobeEmerg function.

Event variable requires the filter enabling for a particular emergencies class. The parameter can have the following values:

Event	Value	Description
RECOVERABLE_EMERGENCY	0x00000001	Acknowledge for recoverable emergency
UNRECOVERABLE_EMERGENCY	0x00000002	Acknowledge for not recoverable emergency
ERROR_BROADCAST	0x00000004	Acknowledge for error notification
WARNING_BROADCAST	0x00000008	Acknowledge for warning notification

Enable requests the notification enabling.

Enable	Value	Description
	TRUE	Enable notification
	FALSE	Disable notification

In this case, if *PlcFiltEnab* is TRUE, also the events required within the PLC task are notified, but the request within the PLC won't be filtered.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150014	Wrong Event parameter

Example:

```
Ret : dword;
Ret := SetFilterEmerg ( RECOVERABLE_EMERGENCY, true) ;
Ret := SetFilterEmerg ( UNRECOVERABLE_EMERGENCY, false) ;
```

See also:

[AckStrobeEmerg](#), [GetFilterEmerg](#), [ClearStrobeEmerg](#), [ReadFilterEmerg](#).

4.37 WarningMessageDisplay

Function `WarningMessageDisplay` displays a series of warning messages by turns

Syntax:

```
ret_code := WarningMessageDisplay (DisplayTime, NumOfMsgs, FirstMsg, OnOffMWIndex, OnHistory,
                                   ActualMsg) ;
```

Input parameters:

- `DisplayTime` (INT) Display time with 10 ms base
- `NumOfMsgs` (INT) Number of messages to be displayed
- `FirstMsg` (INT) Index of first A variable in which the messages are placed
- `OnOffMWIndex` (INT) Activation/Deactivation MW
- `OnHistory` (INT) Request to save message in system history file

Output parameters

- `ActualMsg` (INT) Index of message currently displayed

Execution mode:

Immediate

Use:

This function displays one or more user-defined messages; these messages must be placed in the A variables starting from the one defined by the `FirstMsg` index. If there are several messages to be displayed simultaneously, they are scanned and displayed according to the parameter `DisplayTime`.

Parameter `OnOffMWIndex` defines the zone of the MW variables in which the activation/deactivation signals of the relative message are placed. The first bit of the MW in question (if ON) activates the display of the first message (the one placed in the A variables at index `FirstMsg`), the second bit activates the display of the second message and so on.

The message is displayed on screen and is saved in the system history file if expressly requested by parameter `OnHistory` (value other than 0).



A request to save a message in the System History file may cause the latter to fill up quickly. Use this feature with great caution.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0015000C	Index variable A or MW wrong or out of range

Example:

```
Ret : dword;
Cur : INT ;
(*Displays the messages from A17 until A21 each for 3 seconds by turns *)
MW2000 := 16#1F ;
Ret := WarningMessageDisplay (300, 5, 17, 2000, 0, Cur) ;
```

See also:

[WarningMessage](#)

4.38 TranslateError

The TranslateError function returns the message corresponding to a specific error code.

Syntax:

```
ret_code := TranslateError (TranslationType, Error, ErrorMessage) ;
```

Input Parameters:

<i>TranslationType</i> (INT)	Error type
<i>Error</i> (DWORD)	Error code to translate

Output parameter:

<i>ErrorMessage</i> (STRING)	String with the error description
------------------------------	-----------------------------------

Execution mode:

Immediate

Use:

The function TranslateError returns a message corresponding to an error code. The message generated will contain characters --- when the error to translate needs additional data that can be signalled by this function. *Error* parameter contains the error code: error class plus error. *TranslationType* parameter indicates the type of error translation and can have the flow values:

TranslationType	Mnemonic	Description
0x0000	tr_ERROR	Error
0x0001	tr_LOG	Log
0x0002	tr_EMERGENCY	Emergency
0x0003	tr_ANOMALY	Anomaly

Once the function is finished, *ErrorMessage* parameter will contain the error message calculated from the function.



As the TranslateError execution time is quite long, it would be better not to call this function in a PLC task having high priority.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150098	Wrong <i>TranslationType</i> parameter

Example:

```
Ret : DWORD;
ErrorMessage : STRING ;
Ret := TranslateError (tr_ERROR, 16#00120002, ErrorMessage) ;
```

See also:

TranslateErrorExt

4.39 TranslateErrorExt

The TranslateErrorExt function returns a message corresponding to a specific error code.

Syntax:

```
ret_code := TranslateErrorExt (TranslationType, Error, ArrData, ErrorMessage) ;
```

Input parameters:

<i>TranslationType</i> (INT)	Error type
<i>Error</i> (DWORD)	Error code to translate
<i>ArrData</i> (Array of BYTE)	Additional data related to the error

Output parameter:

<i>ErrorMessage</i> (STRING)	String containing the error description
------------------------------	---

Execution mode:

Immediate

Use:

TranslateErrorExt function returns the message corresponding to a specific error code, with all the additional information given by the system. If all the information is not provided, TranslateErrorExt executes the conversion without including additional information (therefore information will be signalled by the sequence ---). *Error* parameter contains the error code: error class plus error. *TranslationType* parameter indicates the type of error translation and can have the following values:

TranslationType	Mnemonic	Description
0x0000	tr_ERROR	Error
0x0001	tr_LOG	Log
0x0002	tr_EMERGENCY	Emergency
0x0003	tr_ANOMALY	Anomaly

ArrData parameter contains all the additional information related to the error to convert. Data are signalled by the PLC system within the emergency filter tasks; the task is recalled after the emergency notification of an error, of a warning or of an anomaly. Once the function is finished, *ErrorMessage* field will contain the error message calculated from the function.



As the TranslateError execution time is quite long, it would be better not to call this function in a PLC task having high priority.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150098	Wrong <i>TranslationType</i> parameter

Example:

```
Ret           : DWORD;
ErrorMessage : STRING;
Extra         : array [0..147] of BYTE;

Ret := TranslateErrorExt (tr_ERROR, 16#00120002, Extra, ErrorMessage) ;
```

See also:

[TranslateError](#)

4.40 GetBoardInfo

Function GetBoardInfo supplies a description of the resources owned by the axis board.

Syntax:

```
ret_code := GetBoardInfo (Board, Info) ;
```

Input parameters:

Board (INT) Type of Board

Output parameters:

Info (BoardInfo_Struct) Structure of description resources of Board

Execution mode:

Immediate

Use:

Within the *Info structure*, it supplies a description of all the Hardware and Software resources in possession of the axis board. On the basis of the *Board* parameter, it is possible to request the resources associated with the main board (with parameter = 0) as well as a secondary, or additional, resource, if present (parameter other than 0).

The *Info* structure will contain the following information:

Field	Type	Description
BoardID	WORD	Emergency code signalled (see the following table)
HwRevision	DWORD	Number of revision Hardware
FwRevision	DWORD	Number of revision Firmware
BusDescr1	DWORD	First descriptor of Bus
BusDescr2	DWORD	Second descriptor of Bus
BusDescr3	DWORD	Third Bus descriptor
BusDescr4	DWORD	Fourth Bus descriptor
ErdcDescr	DWORD	Descriptor resource Erdc
AdDescr	DWORD	Descriptor resource AD
IoDescr	DWORD	Descriptor resources of IO
MemoryAddr [0..19]	DWORD	Array containing the memory addresses used by the resources
MemorySize [0..19]	DWORD	Array containing the memory dimensions (in bytes) used by the resources
HwCnfDescr	DWORD	Hardware configuration descriptor

Field *BoardID* defines the type of board present and it may assume the following values:

Field	Value	Description
OS_WinCe	1	Emulator on WindowsCE
OS_8182	200	OS8182 board (CanOPEN)
OS_8795	300	4 axes analogic board OS8795
OS_8795_1	301	Secondary 4 axes analogic board OS8795
OS_2005	1000	OS2005 PCI board
OS_2011	1010	OS2011 PCI board
OS_2015	1030	PC104 Master EtherCAT board+ CANOpen
OS_2020	1040	PCI Master EtherCAT board + MLIII + CANOpen

Field *HwRevision* defining the Hardware revision of the board must be read by bits according to the following structure:

Field	bits	Description
FieldCode	0-7	Fixed at 1
HwRevision	8-15	Number of revision Hardware of the board
PLDRevision	16-23	Number of revision of PLD
HwStatus	24-27	Outcome of Hardware test of the board
Reserved	28-31	

Field *FwRevision* defining the revision of the Firmware installed on the board must be read by bits according to the following structure:

Field	bits	Description
FieldCode	0-7	Fixed at 2
SwRevision	8-15	Number of Software revision of the board
NIOSRevision	16-23	Number of revision of system NIOS
SwStatus	24-27	Diagnostic result of boot
FwStatus	28-31	Diagnostic result NIOS

Fields *BusDescr1*, *BusDescr2* and *BusDescr3* defining the communication buses managed by the board, if any, also have a reading structure:

Field	bits	Description
FieldCode	0-7	Fixed at 3
BusType	8-15	Type of Bus 1 (Sercos) 2 (Mechatrolink) 3 (OSWire)
BusRevision	16-23	Number of revision of Bus
Reserved	24-31	

Digital bus types in *BusType* are:

Mnemonic	Value	Description
DGT_BUS_SERCOS	1	SERCOS
DGT_BUS_MECHATROLINK	2	Yaskawa Mechatrolink
DGT_BUS_OSWIRE	3	OS-Wire
DGT_BUS_CANOPEN	4	CANOpen
DGT_BUS_PROFIBUS	8	Profibus
DGT_BUS_ETN	9	ETN
DGT_BUS_ETHERCAT	10	EtherCAT

Field *ErdcDescr* defines the D/As and the Encoders present and it must be read by bits according to the following structure:

Field	bits	Description
FieldCode	0-7	Fixed at 4
HresDAType	8-11	High resolution D/A
HresDANum	12-15	High resolution D/A number
LresDAType	16-19	Low resolution D/A type
LresDANum	20-23	Low resolution D/A number
EncoderType	24-27	Encoder type
EncoderNum	28-31	Encoder number

Bench 1 includes D/A converters from 1 to 8, Bench 2 those 9 to 16.

D/A convertor types are:

Mnemonic	Value	Description
DA_OS20xx	0	OS20xx 16 bit latch HW
-	1-2	Reserved
DA_OS8795	3	OS8795 16 bit latch HW
DA_OS2020	4	OS2020 16 bit latch HW

Encoder types are:

Mnemonic	Value	Description
ENC_OS20xx	0	OS20xx
-	1-2	Reserved
ENC_OS8795	3	OS8795

Field *AdDescr* defines the A/Ds present and it must be read by bits according to the following structure:

Field	bits	Description
FieldCode	0-7	Fixed at 5
Bnk1ADType	8-11	Bench 1 A/D type
Bnk1ADNum	12-15	Bench 1 A/D number
Bnk2ADType	16-19	Bench 2 A/D type
Bnk2ADNum	20-23	Bench 2 A/D number
Bnk1ADMode	24-27	Bench 1 current/voltage AD mode (0/1)
Bnk2ADMode	28-31	Bench 2 current/voltage AD mode (0/1) (0/1)

Bench 1 includes A/D convertors from 1 to 4, bench 2 those from 5 to 8. *ADMode* fields return the Current (bit at 0) or Voltage (bit at 1) setting for every bit referred to a convertor.

A/D convertor types are:

A/D Type	Value	Description
AD_OS20xx	0	OS20xx 12 bit latch HW
-	1-2	Reserved
AD_OS2020	3	OS2020 12 bit latch HW

Field *IoDescr* defines the I/Os present and it must be read by bits according to the following structure:

Field	bits	Description
FieldCode	0-7	Fixed at 6
NumInputs	8-15	Digital inputs number
NumOutputs	16-23	Digital outputs number
NumFastInput	24-27	Fast input number
NumFastOutput	28-31	Fast output number

The *HwCnfDescr* field contains an hardware configuration board descriptor having the following values:

Field	bits	Description
FieldCode	0-7	Fixed at 7
HwCnf1	8-15	Configuration 1
HwCnf2	16-23	Configuration 2
HwCnf3	28-31	Configuration 3

In HwCnf1 there are the following information (at bit):

Mnemonic	bit	Description
EXPORT_VERS	0	Axes limitation for export
-	1-7	Reserved

Return values:

The following table gives the possible values assumed by variable *ret_code*:

<i>ret_code (HEX)</i>	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Info : BoardInfo_Struct;
Ret := GetBoardInfo (0, Info) ;
```

4.41 GetSystemInfo

GetSystemInfo function provides the CNC system type.

Syntax

```
ret_code := GetSystemInfo (Info) ;
```

Output parameter

Info (INT) CNC system type

Execution mode

Immediate

Use

Within the *Info* variable, it provides information defining the system type (CPU) hosting the CNC. The following table lists system codes.

Value	Mnemonic	Description
5	NISE3100	Open M/L
10	EBC_340	Open XS
11	CEWIN	Open with double operating system
12	NISE3140	Open XL

Return value

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example

```
Ret : dword;
Info : INT;
Ret := GetSystemInfo (Info) ;
```

4.42 OsWire_GetEmcyInfo

OsWire_GetEmcyInfo function reads emergency data on I/O nodes connected on the OSWire Bus.

Syntax

```
ret_code := OsWire_GetEmcyInfo (OsWire_Emg) ;
```

Output parameter

OsWire_Emg (OsWireEmg_Struct) Structure where emergency data are read

Execution mode

Immediate

Use

The OSWire I/O nodes status can be monitored as the data can be read through a structure considered as function parameter. The OSWire I/O nodes status can always be monitored to know their status. Data are read within a structure considered as a function parameter.

This must be an OsWireEmg_Struct type structure as follow:

Field	Type	Description
Status	WORD	Error code on the BUS
StatusIO	WORD	Contains the Console, Compact and MiniCompact modulus diagnostics.
NodeID	WORD	Identifies the node displaying an error
StatusOnRIO	BYTE	Defines the error on the RIO Node
ErrorOnRio	BYTE	Defines which modulus connected to the RIO has an error.

In case of recoverable emergency, EmCode field value can be as follow:

EmCode	Value	Description
osw_WDOG	1	Watch dog
osw_OPENSLOT	2	Modulus/Coupler contact released
osw_MAXRETRY	3	Modulus/Coupler communication interrupted
osw_COMMUNICATION	6	BUS communication error

Concerning the StatusIO field, the channels having a possible short circuit are reported. Each 4 I/O group has a channel, therefore on moduluss with 40 I/O there will be 10 channels (max configuration for Console, Compact and MiniCompact).

StatusIO	bit	Description
	0..9	If =1 there is no short circuit, if =0 there is a short circuit
osw_MODWDOG	10	WatchDog on the modulus

StatusOnRIO field displays the errors on a Modulus connected to the RIO coupler.

StatusOnRIO	bit	Description
osw_RIOSHORTCUT	0	If =1 there is an overcharge or a short circuit on the output channel (important signal only if there is the 24V)
osw_RIOPARITY	1	Parity Error on the Coupler/Modulus communication
osw_RIOCOMM	3	Coupler/Modulus communication error
osw_RIOPOWER	4	Moduluss power failure

The ErrorOnRIO field allows to identify on which modulus connected to the coupler there is the error displayed in the previous field. Up to 8 moduluss can be associated to a coupler.

ErrorOnRIO	bit	Description
	0..7	If =1 there is a modulus error. Bit 0 first modulus error, bit 1 second modulus error and so on.

Return values

The following table lists all values *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example

```
Ret      : dword;
OswErr  : OsWireEmg_Struct;
Ret      := OsWire_GetEmcyInfo (OswErr);
```

4.43 ProbeDelete

ProbeDelete function deletes a PLC diagnostic object.

Syntax

```
ret_code := ProbeDelete (ProbeID) ;
```

Input parameter

ProbeID (INT) Probe number [0..255]

Execution mode

Immediate

Use

In the WinPLUS system there are 256 probes allowing to run diagnostic operations (execution times) and to verify synchronism between PLC objects. Diagnostic activities can be detected using the OPEN Oscilloscope system. This function allows deletion of a probe previously created through the ProbeCreate function.

Return value

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0X001700E8	Non-existent Probe

Example

```
Ret : dword;
Ret := ProbeCreate (10, 'MyProbe1') ;
Ret := ProbeDelete (10) ;
```

See also

ProbeCreate, ProbeON e ProbeOFF

4.44 ProbeON

ProbeON function sets Probe signal at ON.

Syntax

```
ret_code := ProbeON (ProbeID) ;
```

Input parameter

ProbeID (INT) Probe number [0..255]

Execution mode

Immediate

Use

In the WinPLUS system there are 256 probes available to run diagnostic operations (execution times) and to verify synchronisation between PLC objects. Diagnostic activities can be detected using the OPEN Oscilloscope system. This function sets the probe created through the ProbeCreate function to ON status.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0X001700E8	Non-existent Probe

Example

```
Ret : dword;
Ret := ProbeCreate (10, 'MyProbe1') ;
Ret := ProbeON (10) ;
...
Ret := ProbeOFF (10) ;
```

See also

ProbeCreate, ProbeDelete e ProbeOFF

4.45 ProbeCreate

ProbeCreate function creates a PLC diagnostic object.

Syntax

```
ret_code := ProbeCreate (ProbeID, ProbeName) ;
```

Input parameters

<i>ProbeID</i> (INT)	Probe number [0..255]
<i>ProbeName</i> (STRING)	Name of the probe

Execution mode

Immediate

Use

In the WinPLUS system there are 256 probes available to run diagnostic operations (execution times) and to verify synchronisation between PLC objects. Diagnostic activities can be detected using the OPEN Oscilloscope system. This function allows creation of a probe; the Probe has to be identified by a *ProbeID* (number from 0 to 255; if < 0 value is 0, if > 255 value is 255) and by a *ProbeName* to be identified by the Oscilloscope.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0X001700E9	Probe already defined
0X001700E6	Too many probe defined

Example

```
Ret : dword;
Ret := ProbeCreate (10, 'MyProbe1') ;
Ret := ProbeON (10) ;
...
Ret := ProbeOFF (10) ;
```

See also

ProbeDelete, ProbeON e ProbeOFF

4.46 ProbeOFF

ProbeOFF function sets Probe signal at OFF.

Syntax

```
ret_code := ProbeOFF (ProbeID) ;
```

Input parameters

ProbeID (INT) Probe number [0..255]

Execution mode

Immediate

Use

In the WinPLUS system there are 256 probes allowing to run diagnostic operations (execution times) and to verify synchronism among PLC object. Diagnostic activities can be detected using the OPEN Oscilloscope system. This function sets on OFF status the probe created through the ProbeCreate function.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0X001700E8	Non-existent probe

Example

```
Ret : dword;
Ret := ProbeCreate (10, 'MyProbe1') ;
Ret := ProbeON (10) ;
...
Ret := ProbeOFF (10) ;
```

See also

ProbeCreate, ProbeDelete e ProbeON

4.47 GetAxisIDFromName

Function GetAxisIDFromName determines the ID of an axis from its name.

Syntax:

```
ret_code := GetAxisIDFromName (Process, AxisName, AxisId) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>AxisName</i> (INT)	Axis name

Output parameters:

<i>AxisId</i> (INT)	Axis identifier [1...32]
---------------------	--------------------------

Execution mode:

Immediate

Use:

This function converts the axis name (in ASCII) belonging to a specified process into a physical identifier of an axis, this is necessary for the main part of the PLC function calls.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150001	Process number wrong
0x00150022	Axis name not found

Example:

```
Ret : dword;
AxId : INT ;
Ret := GetAxisIDFromName (1, AsciiToInt('X'), AxisId) ;
```

See also:

[GetAxisNameFromId](#), [AsciiToInt](#)

4.48 GetAxisNameFromID

Function GetAxisNameFromID determines the name and the process of an axis.

Syntax:

```
ret_code := GetAxisNameFromID (AxisId, Process, AxisName) ;
```

Input parameters:

AxisId (INT) Axis identifier [1...32]

Output parameters:

Process (INT) Process number [1..24]

AxisName (INT) Axis name

Execution mode:

Immediate

Use:

This function converts the axis identifier in the axis name (ASCII) and returns the belonging process number. If the process number is 0 it means that the axis is under control of the machine logic, if it -1 it means that the axis is parked.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00150022	Axis identifier not found
0x00150023	Axis identifier out of range

Example:

```
Ret      : dword;
Proc    : INT ;
AxNamI : INT ;
AxNam  : STRING[1] ;
Ret := GetAxisNameFromID (1, Proc, AxNamI ) ;
AxNam := IntToAscii ( AxNamI ) ;
```

See also:

[GetAxisIDFromName](#), [IntToAscii](#)

4.49 TBL_Lock

Function TBL_Lock requests exclusive access to a table.

Syntax:

```
ret_code := TBL_Lock (Table) ;
```

Input parameters:

Table (INT) Table identifier [0..3]

Execution mode:

Immediate

Use:

This function may be used to prevent other system components that are not WinPLUS from accessing a given table. In this manner, no other user can interfere while the PLC is accessing a table, and therefore this function protects and ensures the integrity of the data contained in it.

Parameter *Table* allows select the table to be protected and it may be:

Table	Value	Description
t_TOOL	0	Tool table
t_OFFSET	1	Offset table
t_ORIGIN	2	Origin table
t_USER	3	User table

 A table should remain in locked status for the shortest time possible, so as to enable the other system components to access it. Otherwise, errors or slowdowns in the system's processing functions may occur.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0019000E	Wrong table number
0x00190011	Table already protected

Example:

```
Ret : dword;
Ret := TBL_Lock (t_OFFSET);
```

See also:

TBL_Unlock, TBL_ReadRecord, TBL_ReadField, TBL_WriteRecord, TBL_WriteField, TBL_SearchField

4.50 TBL_Unlock

Function TBL_Unlock gives exclusive access to a table.

Syntax:

```
ret_code := TBL_Unlock (Table) ;
```

Input parameters:

Table (INT) Table identifier [0..3]

Execution mode:

Immediate

Use:

This function gives access to the table specified in parameter *Table* and other system components.

Parameter *Table* is used to select the table to be unprotected and it may have the following values:

Table	Value	Description
t_TOOL	0	Tool table
t_OFFSET	1	Offset table
t_ORIGIN	2	Origin table
t_USER	3	User table



A table should remain in locked status for the shortest time possible, so as to enable the other system components to access it. Otherwise, errors or slowdowns in the system's processing functions may occur.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0019000E	Wrong table number
0x00190011	Table already protected

Example:

```
Ret : dword;
Ret := TBL_Unlock (t_OFFSET);
```

See also:

TBL_Lock, TBL_ReadRecord, TBL_ReadField, TBL_WriteRecord, TBL_WriteField, TBL_SearchField

4.51 TBL_ReadField

Function TBL_ReadField reads a single field in a table record.

Syntax:

```
ret_code := TBL_ReadField (Table, RecNum, FiedId, DataRead) ;
```

Input parameters:

<i>Table</i> (INT)	Table identifier [0..3]
<i>RecNum</i> (INT)	Number of record [1..n]
<i>FiedId</i> (INT)	Identifier of the field to be read [1..n]

Output parameters:

<i>DataRead</i> (ANY_ELEMENTARY)	Variable in which the data is read
----------------------------------	------------------------------------

Possible overloads:

TBL_ReadField(INT, INT, INT, INT)
TBL_ReadField(INT, INT, INT, WORD)
TBL_ReadField(INT, INT, INT, LREAL)
TBL_ReadField(INT, INT, INT, STRING)

Execution mode:

Immediate

Use:

The function reads a single field (specified by *FiedId*) in a table record *RecNum*. The read variable used must be consistent with type of field read.

The *Table* parameter selects the table to be read, which can be:

Table	Value	Description	Max record number	Value
t_TOOL	0	Tool table	t_TOOL_NREC	250
t_OFFSET	1	Offset table	t_OFFSET_NREC	300
t_ORIGIN	2	Origin table	t_ORIGIN_NREC	100
t_USER	3	User table	t_USER_NREC	100

Depending on the table, the *FiedId* field can be:

t_TOOL

FieldId	Value	Description	Variable type
t_TOOL_NAME	1	Tool name	STRING
t_TOOL_STATUS	2	Tool status	WORD
t_TOOL_LIFETYP	3	Type of life	INT
t_TOOL_MAXLIFE	4	Maximum life	LREAL
t_TOOL_REMLIFE	5	Life remaining	LREAL
t_TOOL_OFFSETNUM	6	Offset tool number	INT
t_TOOL_EXTTYPE	7	Reserved	INT
t_TOOL_EXTIND	8	Reserved	INT

t_TOOL_USELR1	9	Data user LREAL (+1..9 for other fields)	LREAL
t_TOOL_USEIN1	19	Data User INT (+1..9 for other fields)	INT

t_OFFSET

FieldId	Value	Description	Variable type
t_OFFSET_LEN1	1	1st original length of tool	LREAL
t_OFFSET_MAXCH1	2	Max variation in 1st length of tool	LREAL
t_OFFSET_ACTCH1	3	Current variation in 1st length of tool	LREAL
t_OFFSET_LEN2	4	2nd original length of tool	LREAL
t_OFFSET_MAXCH2	5	Max variation in 2nd length of tool	LREAL
t_OFFSET_ACTCH2	6	Current variation in 2nd length of tool	LREAL
t_OFFSET_LEN3	7	3rd original length of tool	LREAL
t_OFFSET_MAXCH3	8	Max variation in 3rd length of tool	LREAL
t_OFFSET_ACTCH3	9	Current variation in 3rd length of tool	LREAL
t_OFFSET_LEN4	10	4th original length of tool	LREAL
t_OFFSET_MAXCH4	11	Max variation in 4th length of tool	LREAL
t_OFFSET_ACTCH4	12	Current variation in 4th length of tool	LREAL
t_OFFSET_LEN5	13	5th original length of tool	LREAL
t_OFFSET_MAXCH5	14	Max variation in 5th length of tool	LREAL
t_OFFSET_ACTCH5	15	Current variation in 5th length of tool	LREAL
t_OFFSET_DIA1	16	1st original diameter tool	LREAL
t_OFFSET_MAXCHD1	17	Max variation in 1st diameter of tool	LREAL
t_OFFSET_ACTCHD1	18	Current variation in 1st diameter of tool	LREAL
t_OFFSET_DIA2	19	2nd original diameter tool	LREAL
t_OFFSET_MAXCHD2	20	Max variation in 2nd diameter of tool	LREAL
t_OFFSET_ACTCHD2	21	Current variation in 2nd diameter of tool	LREAL
T_OFFSET_ORIENT	22	Tool orientation	INT
t_TOOL_EXTTYPE	23	Reserved	INT
t_TOOL_EXTIND	24	Reserved	INT
t_TOOL_USELR1	25	Data user LREAL (+1..9 for other fields)	LREAL
t_TOOL_USEIN1	35	Data User INT (+1..9 for other fields)	INT

t_ORIGIN

FieldId	Value	Description	Variable type
t_ORIGIN_ORIG1	1	1° origin (+1..31 for other fields)	LREAL
t_ORIGIN_EXTTYPE	33	Reserved	INT
t_ORIGIN_EXTIND	44	Reserved	INT

t_USER

FieldId	Value	Description	Variable type
t_USER_VAR1	1	1° variable user (+1..3 for other fields)	LREAL

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0019000E	Number of wrong table
0x0019000F	Number of wrong field
0x00190010	Type of field not congruent with the reading variable
0x0019000C	Error of reading (number of wrong record)

Example:

```
Ret      : dword;
VarUsed : LREAL ;
Ret := TBL_ReadField (t_TOOL, 1, t_TOOL_USELR1+2, VarUsed);
(* reads the third variable user of the table tool record 1 *)
```

See also:

[TBL_Lock](#), [TBL_Unlock](#), [TBL_ReadRecord](#), [TBL_WriteRecord](#), [TBL_WriteField](#), [TBL_SearchField](#)

4.52 TBL_WriteField

Function TBL_WriteField writes a single field in a table record.

Syntax

```
ret_code := TBL_WriteField (Table, RecNum, FieldId, DataWrite) ;
```

Input parameters:

<i>Table</i> (INT)	Table identifier [0..3]
<i>RecNum</i> (INT)	Number of record [1..n]
<i>FieldId</i> (INT)	Identifier of the field to be read [1..n]
<i>DataWrite</i> (ANY_ELEMENTARY)	Variable containing the data to be written

Possible overload:

TBL_WriteField(INT, INT, INT, INT)
TBL_WriteField(INT, INT, INT, WORD)
TBL_WriteField(INT, INT, INT, LREAL)
TBL_WriteField(INT, INT, INT, STRING)

Execution mode:

Immediate

Use:

The function makes it possible to write a single field (specified by *FieldId*) in a table record *RecNum*. The write variable used must be consistent with type of field written.

The *Table* parameter lets you select the table to be written, which can be:

Table	Value	Description	Max record number	Value
t_TOOL	0	Tool table	t_TOOL_NREC	250
t_OFFSET	1	Offset table	t_OFFSET_NREC	300
t_ORIGIN	2	Origin table	t_ORIGIN_NREC	100
t_USER	3	User table	t_USER_NREC	100

Depending on the table, the *FieldId* field can be:

t_TOOL

FieldId	Value	Description	Variable type
t_TOOL_NAME	1	Tool name	STRING
t_TOOL_STATUS	2	Tool status	WORD
t_TOOL_LIFETYP	3	Type of life	INT
t_TOOL_MAXLIFE	4	Max life	LREAL
t_TOOL_REMLIFE	5	Life remaining	LREAL
t_TOOL_OFFSETNUM	6	Corrector number	INT
t_TOOL_EXTTYPE	7	Reserved	INT
t_TOOL_EXTIND	8	Reserved	INT
t_TOOL_USELR1	9	Data user LREAL (+1..9 for other fields)	LREAL
t_TOOL_USEIN1	19	Data User INT (+1..9 for other fields)	INT

t_OFFSET

FieldId	Value	Description	Variable type
t_OFFSET_LEN1	1	1st original length of tool	LREAL
t_OFFSET_MAXCH1	2	Max variation in 1st length of tool	LREAL
t_OFFSET_ACTCH1	3	Current variation in 1st length of tool	LREAL
t_OFFSET_LEN2	4	2nd original length of tool	LREAL
t_OFFSET_MAXCH2	5	Max variation in 2nd length of tool	LREAL
t_OFFSET_ACTCH2	6	Current variation in 2nd length of tool	LREAL
t_OFFSET_LEN3	7	3rd original length of tool	LREAL
t_OFFSET_MAXCH3	8	Max variation in 3rd length of tool	LREAL
t_OFFSET_ACTCH3	9	Current variation in 3rd length of tool	LREAL
t_OFFSET_LEN4	10	4th original length of tool	LREAL
t_OFFSET_MAXCH4	11	Max variation in 4th length of tool	LREAL
t_OFFSET_ACTCH4	12	Current variation in 4th length of tool	LREAL
t_OFFSET_LEN5	13	5th original length of tool	LREAL
t_OFFSET_MAXCH5	14	Max variation in 5th length of tool	LREAL
t_OFFSET_ACTCH5	15	Current variation in 5th length of tool	LREAL
t_OFFSET_DIA1	16	1st original diameter tool	LREAL
t_OFFSET_MAXCHD1	17	Max variation in 1st diameter of tool	LREAL
t_OFFSET_ACTCHD1	18	Current variation in 1st diameter of tool	LREAL
t_OFFSET_DIA2	19	2nd original diameter tool	LREAL
t_OFFSET_MAXCHD2	20	Max variation in 2nd diameter of tool	LREAL
t_OFFSET_ACTCHD2	21	Current variation in 2nd diameter of tool	LREAL
T_OFFSET_ORIENT	22	Tool orientation	INT
t_TOOL_EXTTYPE	23	Reserved	INT
t_TOOL_EXTIND	24	Reserved	INT
t_TOOL_USELR1	25	Data user LREAL (+1..9 for other fields)	LREAL
t_TOOL_USEIN1	35	Data User INT (+1..9 for other fields)	INT

t_ORIGIN

FieldId	Value	Description	Variable type
t_ORIGIN_ORIG1	1	1° origin (+1..31 for other fields)	LREAL
t_ORIGIN_EXTTYPE	33	Reserved	INT
t_ORIGIN_EXTIND	44	Reserved	INT

t_USER

FieldId	Value	Description	Variable type
t_USER_VAR1	1	1° variable user (+1..3 for other fields)	LREAL

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0019000E	Number of wrong table
0x0019000F	Number of wrong field
0x00190010	Type of field not congruent with the reading variable
0x0019000D	Error of writing (number of wrong record)

Example:

```
Ret      : dword;
VarUsed : LREAL ;
VarUser := 17.6 ;
Ret := TBL_WriteField (t_TOOL, 1, t_TOOL_USELR1+2, VarUsed) ;
(*Writes the third variable user of the table tool record 1 *)
```

See also:

[TBL_Lock](#), [TBL_Unlock](#), [TBL_ReadRecord](#), [TBL_WriteRecord](#), [TBL_ReadField](#), [TBL_SearchField](#)

4.53 TBL_ReadRecord

Function TBL_ReadRecord reads a table record in its entirety.

Syntax:

```
ret_code := TBL_ReadRecord (RecNum, Record) ;
```

Input parameters:

RecNum (INT) Number of record to be written [1..n]

Output parameters:

Record (ANY) Record to read

Possible overloads:

TBL_ReadRecord(INT,ToolTable_Struct)
 TBL_ReadRecord(INT,OffsetTable_Struct)
 TBL_ReadRecord(INT,OriginTable_Struct)
 TBL_ReadRecord(INT,UserTable_Struct)

Execution mode:

Immediate

Use:

This function makes it possible to read the entire content of a table record. Table type is determined automatically by WinPLUS according to the structure input to the function. The records are read in the Tool table if the input structure is type ToolTable_Struct, in the Offset table if the input structure is type OffsetTable_Struct, in an Origins table if the input structure is type OriginTable_Struct, from the User table if the input structure is UserTable_Struct.

Structure type	Value	Description	Max record number	Value
ToolTable_Struct	0	Tool table	t_TOOL_NREC	250
OffsetTable_Struct	1	Offset table	t_OFFSET_NREC	300
OriginTable_Struct	2	Origin table	t_ORIGIN_NREC	100
UserTable_Struct	3	User table	t_USER_NREC	100

To be able to ensure the consistency of the data read in the tables, when it proves necessary to read several records or several tables with “cross” data (e.g., read a tool and the relative offsets that are given in separate tables), it is advisable to use the lock/unlock mechanisms supplied by functions TBL_Lock and TBL_Unlock.

The data structures associated with the records of the various tables are described below:

ToolTable_Struct

Field name	Description	Field type
ToolCode	Identifier/Tool name	STRING
Status	Tool status	WORD
LifeType	Type of tool life	INT
MaxLife	Max value of tool life	LREAL
Rem Life	Life remaining	LREAL

OffsNum	Number of associated corrector	INT
ExtType	Reserved	INT
ExtInd	Reserved	INT
dUser	User variables of type LREAL	Array [0..9] of LREAL
sUser	User variables of type INT	Array [0..9] of INT

OffsetTable_Struct

Field name	Description	Field type
InitialLen	Initial length (5 axes)	Array [0..4] of LREAL
MaxChangeLen	Variation max length (5 axes)	Array [0..4] of LREAL
ActChangeLen	Variation current length (5 axes)	Array [0..4] of LREAL
InitialDiam	Initial diameter (2)	Array [0..1] of LREAL
MaxChangeDiam	Variation max diameter (2)	Array [0..1] of LREAL
ActChangeDiam	Variation current diameter (2)	Array [0..1] of LREAL
Orient	Tool orientation	INT
ExtType	Reserved	INT
ExtInd	Reserved	INT
dUser	User variables of type LREAL	Array [0..9] of LREAL
sUser	User variables of type INT	Array [0..9] of INT

OriginTable_Struct

Field name	Description	Field type
OriginValue	Value of origin for 32 axes	Array [0..31] of LREAL
ExtType	Reserved	INT
ExtInd	Reserved	INT

UserTable_Struct

Field name	Description	Field type
UserVal	User variables	Array [0..3] of LREAL

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0019000C	Reading error
0x0019000E	Table type wrong

Example:

```

Ret      : dword;
ToolRc   : ToolTable_struct ;
OffsRc   : OffsetTable_struct ;
Ret := TBL_Lock (t_OFFSET) ;
Ret := TBL_Lock (t_TOOL) ;
Ret := TBL_ReadRecord (1, ToolRc ) ;
Ret := TBL_ReadRecord (ToolRc.OffsNum, OffsRc ) ;
Ret := TBL_Unlock (t_TOOL) ;
Ret := TBL_Unlock (t_OFFSET) ;

```

See also:

[TBL_Lock](#), [TBL_Unlock](#), [TBL_WriteRecord](#), [TBL_WriteField](#), [TBL_ReadField](#), [TBL_SearchField](#)

4.54 TBL_WriteRecord

Function TBL_WriteRecord writes a table record in its entirety

Syntax:

```
ret_code := TBL_WriteRecord (RecNum, Record) ;
```

Input parameters:

RecNum (INT) Number of records to write [1..n]
Record (ANY) Record to be written

Possible overload:

TBL_WriteRecord(INT,ToolTable_Struct)
TBL_WriteRecord(INT,OffsetTable_Struct)
TBL_WriteRecord(INT,OriginTable_Struct)
TBL_WriteRecord(INT,UserTable_Struct)

Execution mode:

Immediate

Use:

This function writes the entire content of a table record. Table type is determined automatically by WinPLUS according to the structure input to the function. The records are written in the Tool table if the input structure is type ToolTable_Struct, in the Offset table if the input structure is type OffsetTable_Struct, in an Origins table if the input structure is type OriginTable_Struct, from the User table if the input structure is UserTable_Struct.

Structure type	Description	Max record number	Value
ToolTable_Struct	Tool table	t_TOOL_NREC	250
OffsetTable_Struct	Offset table	t_OFFSET_NREC	300
OriginTable_Struct	Origin table	t_ORIGIN_NREC	100
UserTable_Struct	User table	t_USER_NREC	100

To ensure the consistency of the data written in the tables, when it proves necessary to write several records or several tables with “cross” data (e.g., write a tool and the relative offsets that are given in separate tables), it is advisable to use the lock/unlock mechanisms supplied by functions TBL_Lock and TBL_Unlock.

The data structures associated with the records of the various tables are described below:

ToolTable_Struct

Field name	Description	Field type
ToolCode	Identifier/Tool name	STRING
Status	Tool status	WORD
LifeType	Type of tool life	INT
MaxLife	Max value of tool life	LREAL
Rem Life	Life remaining	LREAL
OffsNum	Corrector associated number	INT
ExtType	Reserved	INT

ExtInd	Reserved	INT
dUser	Variables user of type LREAL	Array [0..9] of LREAL
sUser	Variables user of type INT	Array [0..9] of INT

OffsetTable_Struct

Field name	Description	Field type
InitialLen	Initial length (5 axes)	Array [0..4] of LREAL
MaxChangeLen	Variation max length (5 axes)	Array [0..4] of LREAL
ActChangeLen	Variation current length (5 axes)	Array [0..4] of LREAL
InitialDiam	Initial diameter (2)	Array [0..1] of LREAL
MaxChangeDiam	Variation max diameter (2)	Array [0..1] of LREAL
ActChangeDiam	Variation current diameter (2)	Array [0..1] of LREAL
Orient	Tool orientation	INT
ExtType	Reserved	INT
ExtInd	Reserved	INT
dUser	Variables user of type LREAL	Array [0..9] of LREAL
sUser	Variables user of type INT	Array [0..9] of INT

OriginTable_Struct

Field name	Description	Field type
OriginValue	Origin value for 32 axes	Array [0..31] of LREAL
ExtType	Reserved	INT
ExtInd	Reserved	INT

UserTable_Struct

Field name	Description	Field type
UserVal	User variables	Array [0..3] of LREAL

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0019000C	Reading error
0x0019000E	Table type wrong

Example:

```

Ret      : dword;
ToolRc   : ToolTable_struct ;
OffsRc   : OffsetTable_struct ;
Ret := TBL_Lock (t_OFFSET) ;
Ret := TBL_Lock (t_TOOL) ;
ToolRc.OffsNum = 5 ;
Ret := TBL_WriteRecord (1, ToolRc );
Ret := TBL_WriteRecord (ToolRc.OffsNum, OffsRc ) ;
Ret := TBL_Unlock (t_TOOL) ;
Ret := TBL_Unlock (t_OFFSET) ;

```

See also:

[TBL_Lock](#), [TBL_Unlock](#), [TBL_ReadRecord](#), [TBL_WriteField](#), [TBL_ReadField](#), [TBL_SearchField](#)

4.55 TBL_SearchField

Function TBL_SearchField makes it possible to search a single field in a table record.

Syntax:

```
ret_code := TBL_SearchField (Table, FiedId, SrcValue, StartRec, StopRec, RecNum) ;
```

Input parameters:

<i>Table</i> (INT)	Table identifier [0..3]
<i>FiedId</i> (INT)	Identifier of the field to be read [1..n]
<i>SrcValue</i> (ANY_ELEMENTARY)	Variable containing the information to search
<i>StartRec</i> (INT)	Search start record [1..n]
<i>StopRec</i> (INT)	Search end record [1..n]

Output parameter:

<i>RecNum</i> (INT)	Number of records in which the field has been found
---------------------	---

Possible overloads:

TBL_SearchField(INT, INT, INT, INT, INT, INT)
TBL_SearchField(INT, INT, WORD, INT, INT, INT)
TBL_SearchField(INT, INT, LREAL, INT, INT, INT)
TBL_SearchField(INT, INT, STRING, INT, INT, INT)

Execution mode:

Immediate

Use:

The function searches a single field (specified by *FiedId*) in a table record from *StartRec* until *StopRec*. The search variable used must be consistent with type of field searched.

The *Table* parameter selects the table to be searched, which can be:

Structure type	Value	Description	Max record number	Value
ToolTable_Struct	0	Tool table	t_TOOL_NREC	250
OffsetTable_Struct	1	Offset table	t_OFFSET_NREC	300
OriginTable_Struct	2	Origin table	t_ORIGIN_NREC	100
UserTable_Struct	3	User table	t_USER_NREC	100

Depending on the table, the *FieldId* field can be:

t_TOOL

FieldId	Value	Description	Variable type
t_TOOL_NAME	1	Tool name	STRING
t_TOOL_STATUS	2	Tool status	WORD
t_TOOL_LIFETYP	3	Type of life	INT
t_TOOL_MAXLIFE	4	Max life	LREAL
t_TOOL_REMLIFE	5	Life remaining	LREAL
t_TOOL_OFFSETNUM	6	Corrector number	INT
t_TOOL_EXTTYPE	7	Reserved	INT
t_TOOL_EXTIND	8	Reserved	INT
t_TOOL_USELR1	9	Data user LREAL (+1..9 for other fields)	LREAL
t_TOOL_USEIN1	19	Data User INT (+1..9 for other fields)	INT

t_OFFSET

FieldId	Value	Description	Variable type
t_OFFSET_LEN1	1	1st original length of tool	LREAL
t_OFFSET_MAXCH1	2	Max variation in 1st length of tool	LREAL
t_OFFSET_ACTCH1	3	Current variation in 1st length of tool	LREAL
t_OFFSET_LEN2	4	2nd original length of tool	LREAL
t_OFFSET_MAXCH2	5	Max variation in 2nd length of tool	LREAL
t_OFFSET_ACTCH2	6	Current variation in 2nd length of tool	LREAL
t_OFFSET_LEN3	7	3rd original length of tool	LREAL
t_OFFSET_MAXCH3	8	Max variation in 3rd length of tool	LREAL
t_OFFSET_ACTCH3	9	Current variation in 3rd length of tool	LREAL
t_OFFSET_LEN4	10	4th original length of tool	LREAL
t_OFFSET_MAXCH4	11	Max variation in 4th length of tool	LREAL
t_OFFSET_ACTCH4	12	Current variation in 4th length of tool	LREAL
t_OFFSET_LEN5	13	5th original length of tool	LREAL
t_OFFSET_MAXCH5	14	Max variation in 5th length of tool	LREAL
t_OFFSET_ACTCH5	15	Current variation in 5th length of tool	LREAL
t_OFFSET_DIA1	16	1st original diameter tool	LREAL
t_OFFSET_MAXCHD1	17	Max variation in 1st diameter of tool	LREAL
t_OFFSET_ACTCHD1	18	Current variation in 1st diameter of tool	LREAL
t_OFFSET_DIA2	19	2nd original diameter tool	LREAL
t_OFFSET_MAXCHD2	20	Max variation in 2nd diameter of tool	LREAL
t_OFFSET_ACTCHD2	21	Current variation in 2nd diameter of tool	LREAL
T_OFFSET_ORIENT	22	Tool orientation	INT
t_TOOL_EXTTYPE	23	Reserved	INT
t_TOOL_EXTIND	24	Reserved	INT
t_TOOL_USELR1	25	Data user LREAL (+1..9 for other fields)	LREAL
t_TOOL_USEIN1	35	Data User INT (+1..9 for other fields)	INT

t_ORIGIN

FieldId	Value	Description	Variable type
t_ORIGIN_ORIG1	1	1° origin (+1..31 for other fields)	LREAL
t_ORIGIN_EXTTYPE	33	Reserved	INT
t_ORIGIN_EXTIND	44	Reserved	INT

t_USER

FieldId	Value	Description	Variable type
t_USER_VAR1	1	1° variable user (+1..3 for other fields)	LREAL

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0019000E	Number of wrong table
0x0019000F	Number of wrong field or not found
0x00190010	Field type not congruent with the writing variable
0x0019000C	Reading error (number of wrong record)

Example:

```

Ret      : dword;
VarUsed : LREAL ;
RecNo   : INT ;
VarUser := 17.6 ;
Ret := TBL_SearchField (t_TOOL, t_TOOL_USELR1+2, 7, 10, VarUsed , RecNo);
(* Search for value 17.6 in the third user variable of the tool table from
record 7 through record 10 *)

```

See also:

[TBL_Lock](#), [TBL_Unlock](#), [TBL_ReadRecord](#), [TBL_WriteRecord](#), [TBL_ReadField](#), [TBL_WriteField](#)

4.56 ConsoleOFF

The function ConsoleOFF disconnects an Operator Console from a process.

Syntax:

```
ret_code := ConsoleOFF (Process, NetType, NodeID) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>NetType</i> (WORD)	Bus on which is connected the console
<i>NodeID</i> (WORD)	Node number

Execution mode:

Immediate

Use:

This function disconnects an Operator Console from the process in question (connected on Bus indicated from NetType with node number NodeID). *NetType* can take on the following values:

NetType	Value	Description
NET_CANOPEN	1	Console on CANOpen Bus
NET_OSWIRE	2	Console on OSWire Bus
NET_EHERCAT	7	Console EtherCat

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00120004	Process number wrong
0x00120007	No Console configured for the process in question
0x0012000F	Value NetType wrong
0x0012000E	Console non-existent
0x00120003	Console not found for Node Nodeld

Example:

```
Ret : dword;
Ret := ConsoleOFF (1, NET_EHERCAT, 3);
```

See also:

ConsoleON, ConsoleSetup, ConsoleReadInput, ConsoleReadOutput, ConsoleWriteOutput,
ConsoleSetupArray

4.57 ConsoleON

The function ConsoleON connects an operator console to a process.

Syntax:

```
ret_code := ConsoleON (Process, NetType, NodeID) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>NetType</i> (WORD)	Bus on which is connected the console
<i>NodeID</i> (WORD)	Node number

Execution mode:

Immediate

Use:

This function activates an Operator Console for the process in question (connected on Bus indicated from NetType with node number NodeID). *NetType* can take on the following values:

NetType	Value	Description
NET_CANOPEN	1	Console on CANOpen Bus
NET_OSWIRE	2	Console on OSWire Bus
NET_EHERCAT	7	Console EtherCat

The I/O configurator utility makes it possible to configure up to 16 operator consoles. Each of these operator consoles has a unique identification code (which is defined by means of the rotary switch of the device). Using the CONS_ON and CONS_OFF functions, different associations between the operator console and the processes can be defined.

The following association criteria must be complied with:

- ▷ Any operator console available can be associated with any available process. Up to 4 operator consoles per process can be active at the same time.
- ▷ Connecting a new operator console to a process which is already associated with 4 consoles, the function will notify an error.
- ▷ An operator console can be assigned simultaneously to up to 4 processes, enabling common operations to be performed on all the processes associated with it.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00120004	Process number wrong
0x00120005	Process already associated with 4 console
0x00120006	Console already associated with the process
0x0012000F	Value NetType wrong
0x0012000E	Console non-existent
0x00120003	Console not found for Node NodId

Example:

```
Ret : dword;  
Ret := ConsoleON (1, NET_ETHERCAT, 3);
```

See also:

[ConsoleOFF](#), [ConsoleSetup](#), [ConsoleReadInput](#), [ConsoleReadOutput](#), [ConsoleWriteOutput](#), [ConsoleSetupArray](#)

4.58 ConsoleReadInput

Function `ConsoleReadInput` reads the status of console inputs.

Syntax:

```
ret_code := ConsoleReadInput (NetType, NodeID, InpData) ;
```

Input parameters:

`NetType` (WORD) Bus on which is connected the console
`NodeID` (WORD) Node number

Output parameters:

`InpData` (DWORD) DWORD where the inputs associated with the console can be read

Execution mode:

Immediate

Use:

This function reads the value of the input signals going to an operator console connected to the bus specified by `NetTyp` with node number `NodeID`. The value of `NetType` may be:

NetType	Value	Description
NET_CANOPEN	1	Console on CANOpen Bus
NET_OSWIRE	2	Console on OSWire Bus
NET_ETHERCAT	7	Console EtherCat

Within the `InpData` variable, inputs (ON means pressed button, 0 button released) are defined as follow:

Signal	Value (hex)	Description
PUSHB_AUTO	0x00000001	AUTOs mode button
PUSHB_MDI	0x00000002	MDIs mode button
PUSHB_BLKBLK	0x00000004	Execution in STEPs mode button
PUSHB_JOGINC	0x00000008	Jog Incrementals mode button
PUSHB_HOME	0x00000010	Homing mode button
PUSHB_MANUAL	0x00000020	Manuals mode button
PUSHB_RESET	0x00000040	Resets button
PUSHB_CYCLE	0x00000080	Cycles button
PUSHB_HOLD	0x00000100	Holds button
PUSHB_JOGDIRPOS	0x00000200	Positive JogDir button
PUSHB_JOGDIRNEG	0x00000400	Negative JogDirs button
PUSHB_JOGCYCPOS	0x00000800	Positive JogDir button + Cycle
PUSHB_JOGCYCNEG	0x00001000	Negative JogDir button + Cycle
PUSHB_RET PROF	0x00002000	Return on the profile modality button
PUSHB_HPG	0x00004000	HPG modality button (handwheel)
PUSHB_P1	0x00010000	P1 button
PUSHB_P2	0x00020000	P2 button
PUSHB_P3	0x00040000	P3 button
PUSHB_P4	0x00080000	P4 button

PUSHB_P5	0x00100000	P5 button
PUSHB_P6	0x00200000	P6 button



The buttons P1-P6, JOGDIR and JOGCYC RETPROF and HPG are available only for NET_ETHERCAT console.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0012000F	Value NetType wrong
0x0012000E	Console non-existent
0x00120003	Console not found for Node Nodeld

Example:

```
Ret : dword;
InpV : DWORD ;
Ret := ConsoleReadInput (NET_ETHERCAT, 3, OutV);
```

See also:

[ConsoleON](#), [ConsoleOFF](#), [ConsoleSetup](#), [ConsoleWriteOutput](#), [ConsoleReadOutput](#), [ConsoleSetupArray](#)

4.59 ConsoleReadOutput

Function ConsoleReadOutput reads the status of console outputs.

Syntax:

```
ret_code := ConsoleReadOutput (NetType, NodeID, OutData) ;
```

Input parameters:

NetType (WORD) Bus on which is connected the console
NodeID (WORD) Node number

Output parameters:

OutData (ANY_ELEMENTARY) Outputs read status

Execution mode:

Immediate

Use:

This function reads the value of the output signals going to an operator console connected to the bus specified by *NetType* with node number *NodeID*. The value of *NetType* may be:

NetType	Value	Description
NET_CANOPEN	1	Console on CANOpen Bus
NET_OSWIRE	2	Console on OSWire Bus
NET_ETHERCAT	7	Console EtherCat

In the *OutData* variable, output signals are defined as follows:

Signal	Value (hex)	Description
LED_AUTO	0x00000001	Led associated to AUTO mode
LED_MDI	0x00000002	Led associated to MDI mode
LED_BLKBLK	0x00000004	Led associated to execution mode in STEP
LED_JOGINC	0x00000008	Led associated to Jog Incremental mode
LED_HOME	0x00000010	Led associated to Homing mode
LED_MANUAL	0x00000020	Led associated to Manual mode
LED_RESET	0x00000040	Led associated to Reset status
LED_CYCLE	0x00000080	Led associated to Cycle status
LED_HOLD	0x00000100	Led associated to Hold status
LED_JOGDIRPOS	0x00010000	Led associated to JOGDIR+ button
LED_JOGDIRNEG	0x00020000	Led associated to JOGDIR- button
LED_JOGCYCPOS	0x00040000	Led associated to JOGDIR+ + Cycle button
LED_JOGCYCNEG	0x00080000	Led associated to JOGDIR- + Cycle button
LED_RETPROF	0x00100000	Led associated to the return status on the profile
LED_HPG	0x00200000	Led associated to the HPG status (hadwheel)
LED_P1	0x00000001	Led associated to P1 button
LED_P2	0x00000002	Led associated to P2 button
LED_P3	0x00000004	Led associated to P3 button
LED_P4	0x00000008	Led associated to P4 button

Signal	Value (hex)	Description
LED_P5	0x00000010	Led associated to P5 button
LED_P6	0x00000020	Led associated to P6 button



Leds P1-P6, JOGDIR and JOGCYC RETPROF and HPG are available only for NET_ETHERCAT console.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0012000F	Value NetType wrong
0x0012000E	Console non-existent
0x00120003	Console not found for Node Nodeld

Example:

```
Ret : dword;
OutV : WORD ;
Ret := ConsoleReadOutput (NET_ETHERCAT, 3, OutV);
OutV := OutV or LED_P1;
Ret := ConsoleWriteOutput (NET_ETHERCAT, 3, OutV);
```

See also:

[ConsoleON](#), [ConsoleOFF](#), [ConsoleSetup](#), [ConsoleReadInput](#), [ConsoleReadOutput](#), [ConsoleSetupArray](#)

4.60 ConsoleWriteOutput

Function ConsoleWriteOutput writes the status of the Outputs of the console.

Syntax:

```
ret_code := ConsoleWriteOutput (NetType, NodeID, OutData) ;
```

Input parameters:

NetType (WORD) Bus on which the console is connected

NodeID (WORD) Node number

OutData (ANY ELEMENTARY) Outputs status to write

Possible overloads

ConsoleWriteOutput (WORD,WORD,WORD)

ConsoleWriteOutput (WORD,WORD,DWORD)

Execution mode:

Immediate

Use:

This function writes the values of the output signals going to an operator console connected to the Bus specified by *NetTyp* with node number *NodeID*. The value of *NetType* may be:

NetType	Value	Description
NET_CANOPEN	1	Console on CANOpen Bus
NET_OSWIRE	2	Console on OSWire Bus
NET_ETHERCAT	7	Console EtherCat

In the *OutData* variable, the output signals are defined as follows:

Signal	Value (hex)	Description
LED_AUTO	0x00000001	Led associated to AUTO mode
LED_MDI	0x00000002	Led associated to MDI mode
LED_BLKBLK	0x00000004	Led associated to execution mode in STEP
LED_JOGINC	0x00000008	Led associated to Jog Incremental mode
LED_HOME	0x00000010	Led associated to Homing mode
LED_MANUAL	0x00000020	Led associated to Manual mode
LED_RESET	0x00000040	Led associated to Reset status
LED_CYCLE	0x00000080	Led associated to Cycle status
LED_HOLD	0x00000100	Led associated to Hold status
LED_JOGDIRPOS	0x00000200	Led associated to JOGDIR+ button
LED_JOGDIRNEG	0x00000400	Led associated to JOGDIR- button
LED_JOGCYCPOS	0x00000800	Led associated to JOGDIR+ button + Cycle
LED_JOGCYCNEG	0x00001000	Led associated to JOGDIR- button + Cycle
LED_RETPROF	0x00002000	Led associated to the return on profile status
LED_HPG	0x00004000	Led associated to the HPG status (volantino)
LED_P1	0x00010000	Led associated to P1 button
LED_P2	0x00020000	Led associated to P2 button
LED_P3	0x00040000	Led associated to P3 button

Signal	Value (hex)	Description
LED_P4	0x00080000	Led associated to P4 button
LED_P5	0x00100000	Led associated to P5 button
LED_P6	0x00200000	Led associated to P6 button



Leds P1-P6, JOGDIR and JOGCYC RETPROF and HPG are available only for NET_ETHERCAT console.

Return values:

The following table lists all the possible values that *ret_code* can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0012000F	Value NetType wrong
0x0012000E	Console non-existent
0x00120003	Console not found for Node NodId

Example:

```
Ret : dword;
OutV : WORD ;
Ret := ConsoleReadOutput (NET_ETHERCAT, 3, OutV);
```

See also:

[ConsoleON](#), [ConsoleOFF](#), [ConsoleSetup](#), [ConsoleReadInput](#), [ConsoleWriteOutput](#), [ConsoleSetupArray](#)

4.61 ConsoleSetup

Function ConsoleSetup configures the console selectors for a process in continuous mode.

Syntax:

```
ret_code := ConsoleSetup (Process, Selector, NumOfStep, StepData) ;
```

Input parameters:

<i>Process</i> (INT)	Process number [1..24]
<i>Selector</i> (WORD)	Selector to be configured
<i>NumOfStep</i> (WORD)	Step number to be associated to selector
<i>StepData</i> (DWORD)	Step value

Execution mode:

Immediate

Use:

This function is used to configure the operating modes of selectors, i.e., it defines the variation percentages of SPEED, FEED, RAPID, JOGFEED and JOGINCR to be applied during selector rotation. These configurations apply to the selectors of all the consoles associated with a process. By indicating the process with ALL_PROCESS (-1), the parameters are configured for all the processes to which consoles have been associated.

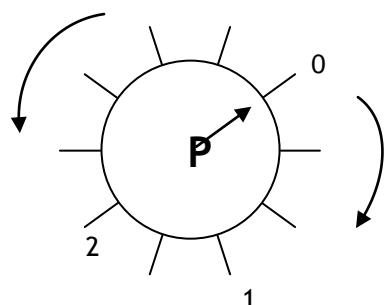
For the consoles that have no selectors (e.g., oplink) and have buttons instead to vary the SPEED, FEED,... percentages, each pressure on a button is regarded as an activation of a “virtual selector” and the direction is given by the +/- pressure on the button.

The *Selector* parameter defines the selector to be configured; it may be:

Selector	Value	Description
ALL_SELECTORS	0xFFFF	All five selectors
SPEED_SELECTOR	0x0000	Selector Speed %
FEED_SELECTOR	0x0001	Selector Feed %
MANFEED_SELECTOR	0x0002	Selector JogFeed %
JOGINCR_SELECTOR	0x0003	Selector JogIncr
RAPID_SELECTOR	0x0004	Selector Rapid %

NumOfStep indicates the number of divisions of the 360° angle of a selector; if this number is assigned with value DEFAULT_NUM_STEP (0), default parameters are used, i.e., 10 steps are used for all selectors save for JogIncr, which is subdivided into 6 steps. This value may vary from MIN_STEP_NUM (2) to MAX_STEP_NUM (51).

For Encoder type selectors, the optimal value for the parameter is a multiple/submultiple of the number of steps of the Encoder. The figure below shows a selector P and the division of the 360° angle into 10 steps (default value). To vary the percentage value of selector speed, it is necessary to perform an angular movement greater than a step of the 360° angle.



StepData defines the value of the individual step (variation %); if this value is assigned with **DEFAULT_VAL_STEP** (0), default parameters are used, i.e., the minimum and maximum range of the selector is divided by the number of steps defined with *NumOfStep*. This parameter is defined in 0.01% units (a value of 1000 corresponds to 10%) and it cannot exceed **MAX_STEP_VAL** (100000 i.e., 1000%).

The default configuration is:

Selector	N° step	Minimum value	Maximum value	Step value
SPEED_SELECTOR	10	7500 - 75%	12500 - 125%	500 - 5%
FEED_SELECTOR	10	0 - 0%	12500 - 125%	1250 - 12,5 %
MANFEED_SELECTOR	10	0 - 0%	10000 - 100%	1000 - 10%
JOGINCR_SELECTOR	6	0 - 0mm	100000 - 10mm	See below
RAPID_SELECTOR	10	0 - 0%	10000 - 100%	1000 - 10%

JogIncr default values:

0.000	0.001	0.010	0.100	1.000	10.000
-------	-------	-------	-------	-------	--------

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00120004	Process number wrong
0x00120009	Value Selector wrong
0x0012000A	Step number not admitted
0x0012000B	Step value wrong

Example:

This example shows how to divide the 360° angle of the selector and how to assign a fixed value to each step.

```
Ret : dword;
Ret := ConsoleSetup(1, SPEED_SELECTOR, 10, DEFAULT_VAL_STEP);
```

Function `ConsoleSetup` is used to set the operation of the SPEED selector of process 1. The 360° angle of the selector is divided into 10 steps (`NumOfStep` = 10) and the value attributed to each of them is 10% (`StepData` = 1000). Having set the values, let us see how speed percentage variations are obtained with reference to the previous figure.

Let us assume that the selector is in position 0. If the selector is rotated from position 0 to position 1 in the clockwise direction (108 degrees), the position is incremented by 3 steps and hence the system's percentage value will be incremented by:

$$3 \times StepData = 30\%$$

If the selector is turned from position 0 to position 2 in the counter-clockwise direction (180 degrees), the position is reduced by 5 steps and hence the system's percentage value will be reduced by:

$$5 \times StepData = 50\%.$$

Hence, incremental/decremented values are fixed and are determined on the basis of the value of `StepData`.

See also:

`ConsoleOFF`, `ConsoleON`, `ConsoleReadInput`, `ConsoleReadOutput`, `ConsoleWriteOutput`,
`ConsoleSetupArray`

4.62 ConsoleSetupArray

Function `ConsoleSetupArray` configures the operating modes of selectors.

Syntax:

```
ret_code := ConsoleSetupArray (Process, Selector, NumOfStep, StepData) ;
```

Input parameters:

<code>Process</code> (INT)	Process number [1..24]
<code>Selector</code> (WORD)	Selector to be configured
<code>NumOfStep</code> (WORD)	Step number to be associated to selector
<code>Mode</code> (WORD)	Array definition mode
<code>StepData</code> (ARRAY of DWORD)	Step value

Execution mode:

Immediate

Use:

This function is used to configure the operating modes of selectors, i.e., it defines the variation percentages of SPEED, FEED, RAPID, JOGFEED and JOGINCR to be applied during selector rotation. These configurations apply to the selectors of all the consoles associated with a process. By indicating the process with ALL_PROCESS (-1), the parameters are configured for all the processes to which consoles have been associated.

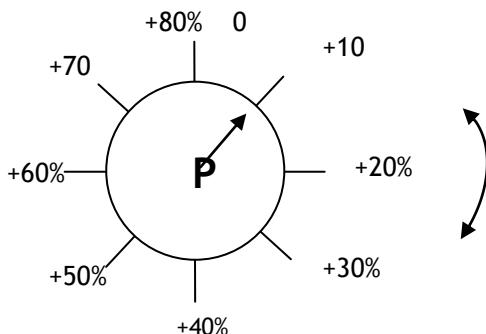
For the consoles that have no selectors (e.g., oplink) and have buttons instead to vary the SPEED, FEED,... percentages, each pressure on a button is regarded as an activation of a “virtual selector” and the direction is given by the +/- pressure on the button.

The `Selector` parameter defines the selector to be configured; it may be:

Selector	Value	Description
ALL_SELECTORS	0xFFFF	All five selectors
SPEED_SELECTOR	0x0000	Selector Speed %
FEED_SELECTOR	0x0001	Selector Feed %
MANFEED_SELECTOR	0x0002	Selector JogFeed %
JOGINCR_SELECTOR	0x0003	Selector JogIncr
RAPID_SELECTOR	0x0004	Selector Rapid %

`NumOfStep` indicates the number of elements in the `StepData` array that defines the values to be assumed by the selectors during their variation; this value subdivides the 360° angle into `NumOfStep`-1 steps.

This value may vary from `MIN_STEP_NUM` (2) to `MAX_STEP_NUM` (51). For Encoder type selectors, the optimal value for the parameter is a multiple/submultiple of the number of steps of the Encoder. The figure below shows a selector P and the division of the 360° angle into 8 steps (9 default values). To vary the percentage value of selector speed, it is necessary to perform an angular movement greater than a step of the 360° angle.



A selector may be configured to have only positive, or only negative or mixed percentages. To this end we have the *Mode* parameter which may assume the following values:

Mode	Value	Description
AR_POSITIVE	0x0000	Selector with only negative %s, that applies to all types of selector
AR_NEGATIVE	0x0001	Selector with only negative %s, that applies only to MANFEED_SELECTOR
AX_MIXED	0x0002	Selector with positive and negative %s, that applies only to MANFEED_SELECTOR
ARR_JOGDIR	0x0003	Selector with % maintaining the direction set by the system via JOGDIR (affects only the value, but not the direction) applicable only for the selector MANFEED_SELECTOR

StepData is used to define the array containing the values (i.e., the variation %s) to be assumed by a selector in its various positions. To have the selector assume the values shown in the previous figure, input an array of the type:

0	1000	2000	3000	4000	5000	6000	7000	8000
---	------	------	------	------	------	------	------	------

The data contained in the array must be congruent with the *Mode* value, i.e., increasing for type AR_POSITIVE and ARR_JOGDIR arrays, decreasing for type AR_NEGATIVE arrays, first decreasing and then increasing for type AR_MIXED arrays; in the latter case, an element containing value 0 MUST be present.

Return values:

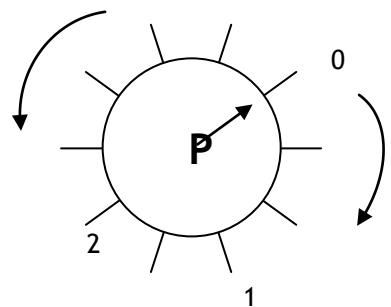
ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00120004	Process number wrong
0x00120009	Selector value wrong
0x0012000A	Step number not admitted
0x0012000C	Values in Array not monotone
0x0012000D	Mode value not applicable to selector

Example 1:

This example shows how to divide the 360° angle of the selector with values not linearly distributed. Assuming we have an array consisting of:

1000	1500	3500	4000	6000	6500	7000	8500	10000	12500
------	------	------	------	------	------	------	------	-------	-------

```
Ret : dword;
Arr : array [0..9] of DWORD ;
Ret := ConsoleSetupArray (1, SPEED_SELECTOR, 10, AR_POSITIVE, Arr) ;
```



Function `ConsoleSetupArray` is used to set the operation of the SPEED selector of process 1. The 360° angle of the selector is divided into 9 steps (`NumOfStep = 10`) and the percentage value is taken from the `StepData` values array.

Let us assume that the selector is in position 0 and the system value is 65%. If the selector is rotated from position 0 to position 1, the position is incremented by 3 steps and hence first of all the drive shall search for the value of the system in the array (65%) and then it shall actualise the next third value (100%).

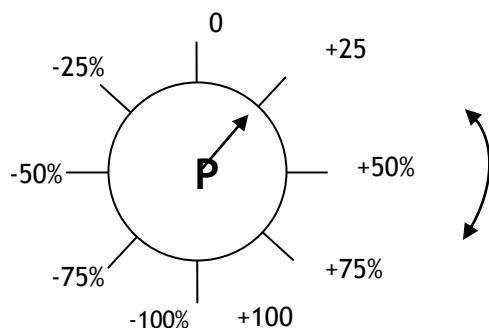
If the selector is turned from position 0 to position 2 in the counter-clockwise direction (180 degrees), the position is reduced by 5 steps and hence the driver will search the system value before inside array (65%) and then it will focus on the previous fifth value (10%)

Example 2:

This example shows how to set the data to have the JOG MANUAL selector theoretically subdivided into two parts, with four variations in the clockwise direction of rotation and four in the counter-clockwise direction (two-directional JOG MANUAL). Assuming we have an array consisting of:

10000	7500	5000	2500	0	2500	5000	7500	10000
-------	------	------	------	---	------	------	------	-------

```
Ret : dword;
Arr : array [0..8] of DWORD ;
Ret := ConsoleSetupArray (1, MANFEED_SELECTOR, 9, AR_MIXED, Arr) ;
```

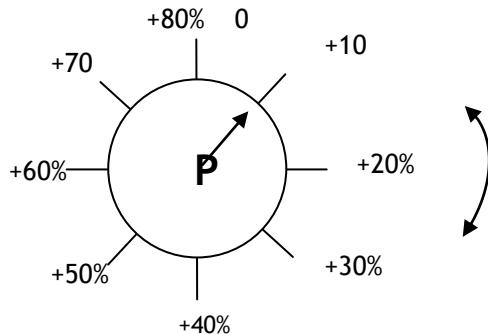


Function `ConsoleSetupArray` is used to set the operation of the selector. The 360° angle of the selector is divided into 8 steps (`NumOfStep = 9`) and the percentage value is taken from the `StepData` values array. The values of the negative part must be placed to the left of the zero, the values of the positive part must go to the right.

For a **uni-directional JOG MANUAL with positive values** (i.e. creating only positive movements directions); the array must consist of:

0	1000	2000	3000	4000	5000	6000	7000	8000
---	------	------	------	------	------	------	------	------

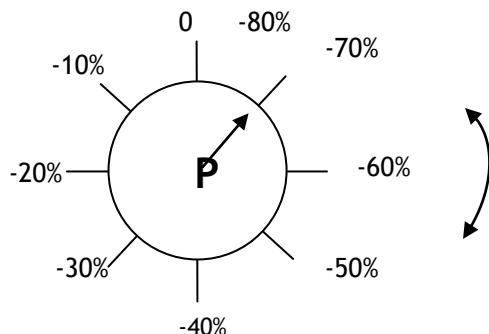
```
Ret : dword;
Arr : array [0..8] of DWORD ;
Ret := ConsoleSetupArray (1, MANFEED_SELECTOR, 9, AR_POSITIVE, Arr) ;
```



It is possible to have a **multi-directional JOG MANUAL** (that means creating only movement requests with the direction selected by the JOGDIR value set by the system), therefore an array must be defined as follows:

8000	7000	6000	5000	4000	3000	2000	1000	0
------	------	------	------	------	------	------	------	---

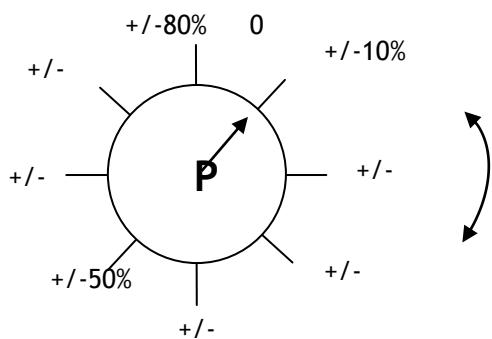
```
Ret : dword;
Arr : array [0..8] of DWORD ;
Ret := ConsoleSetupArray (1, MANFEED_SELECTOR, 9, AR_NEGATIVE, Arr) ;
```



It is possible to have a **multi-directional JOG MANUAL pluri-direzionale** (that means creating only movement requests **always** with direction referred to the JOGDIR value set by the system), therefore it is necessary an array as follow:

0	1000	2000	3000	4000	5000	6000	7000	8000
---	------	------	------	------	------	------	------	------

```
Ret : dword;
Arr : array [0..8] of DWORD ;
Ret := ConsoleSetupArray (1, MANFEED_SELECTOR, 9, ARR_JOGDIR, Arr) ;
```



See also:

[ConsoleOFF](#), [ConsoleON](#), [ConsoleReadInput](#), [ConsoleReadOutput](#), [ConsoleWriteOutput](#), [ConsoleSetup](#)

4.63 Filter_Acc_Create

Filter_Acc_Create function creates a filter for the centripetal acceleration.

Syntax

```
ret_code := Filter_Acc_Create (Kv, Ts, VffPerc, FilterID) ;
```

Input parameters

<i>Kv</i> (LREAL)	Position loop gain
<i>Ts</i> (LREAL)	Sampling time(ms)
<i>VffPerc</i> (LREAL)	vff% used in the motion

Output parameters

<i>FilterID</i> (INT)	Filter Id
-----------------------	-----------

Execution mode

Wait

Use

A typical error occurring during a circular interpolation is the reduction of the measured ray compared to the programmed one. Centripetal acceleration offset filter balances the following error as to behave similarly to the Velocity Feed Forward application, avoiding to make the tool machine too “nervous”.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0001	Maximum number of filters finished

Example

```
Ret : dword;
Kv : lreal := 0.9;
Ts : lreal := 4.0;
VffPerc : lreal := 100.0;
FilterID : int;
Ret := Filter_Acc_Create (Kv, Ts, VffPerc, FilterID);
```

Creates an acceleration with Kv= 0.9, Ts= 2ms and VffPerc = 100%.

See also

[Filter_Acc_Delete](#), [Filter_Acc_Run](#)

4.64 Filter_Acc_Run

Filter_Acc_Run function applies an acceleration filter.

Syntax

```
ret_code := Filter_Acc_Run (FilterID, Command, InpStruct, NPnt, OutStruct) ;
```

Input parameters

<i>FilterID</i> (INT)	Filter identifier
<i>Command</i> (WORD)	Command to execute
<i>InpStruct</i> (Filter_Struct)	Structure of the elements to filter

Output parameters

<i>NPnt</i> (INT)	Number of points still present when Command = FILTER_STOP
<i>OutStruct</i> (Filter_Struct)	Structure of the elements filtered

Execution mode

Immediate

Use

This function applies, re-initialize or stop (see *Command* table) an acceleration filter created before the elements shown in the *Filter_Struct* structure below.

Campo	Description
ElementsNumber	Indicates the number of elements to filter
Elements	Array containing the elements to filter

Command	Description
FILTER_RUN	Runs the filter
FILTER_INIT	Calculates the new starting conditions of the filter
FILTER_STOP	Stops the filter

FILTER_INIT parameter reloads new starting conditions on the filter. The filter is stopped and re-initialized only if called before the FILTER_STOP command. FILTER_STOP parameter allows to stop the filter execution. The application keeps on calling the function until the *NPnt* output (showing the number of points to still present) becomes =0.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>FilterID</i> non-configured

Example

```
Ret : dword;
Kv : int := 0.9;
FilterID : int;
NPnt : int;

(* ---- Creates the filter with Kv = 0.9 ---- *)
Ret := Filter_Acc_Create (Kv, FilterID);

InpStruct: Filter_Struct;
OutStruct: Filter_Struct;
InpStruct.ElementsNumber := 1;
InpStruct.Elements[0] := errors;

(* ---- Applies the filter to the position error ---- *)
Ret := Filter_Acc_Run (FilterID, FILTER_RUN, InpStruct, NPnt, OutStruct);
```

It applies the ID 1 filter to the position error.

See also

[Filter_Acc_Create](#), [Filter_Acc_Delete](#)

4.65 Filter_Bessel_Delete

The Filter_Bessel_Delete function deletes a Bessel filter.

Syntax

```
ret_code := Filter_Bessel_Delete (FilterID) ;
```

Input parameter

FilterID (INT) Filter identifier

Execution mode

Wait

Use

This function deletes a Bessel filter.

Return values

The following table lists all values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>FilterID</i> non-existent

Example

```
Ret : dword;
FilterID : int := 1;
Ret := Filter_Bessel_Delete (FilterID);
```

Deletes a filter with ID1.

See also

[Filter_Bessel_Create](#), [Filter_Bessel_Run](#)

4.66 Filter_Boost_Create

The Filter_Boost_Create function creates a start or reversal filter

Syntax

```
ret_code := Filter_Boost_Create (Tr, Tf, Boost, FilterID) ;
```

Input parameters

<i>Tr</i> (LREAL)	Raise time constant for the first order filter (ms)
<i>Tf</i> (LREAL)	Fall time constant for the first order filter (ms)
<i>Boost</i> (LREAL)	Position boost to recover (mm/inches/degrees)

Output parameters

<i>FilterID</i> (INT)	Filter identifier
-----------------------	-------------------

Execution mode

Wait

Use

This filter restores the “elastic backlash” that happens at reversal or start of axis motion, due to the presence of the non-linear friction on the motor

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0001	Maximum filters number expired

Example

```
Ret : dword;
Tr : lreal := 20.0;
Tf : lreal := 20.0;
Boost : lreal := 2.2;
FilterID : int;
Ret := Filter_Boost_Create (Tr, Tf, Boost, FilterID);
```

See also

[Filter_Boost_Delete](#), [Filter_Boost_Run](#)

4.67 Filter_Boost_Run

The Filter_Boost_Run function runs a start or reversal filter.

Syntax

```
ret_code := Filter_Boost_Run (FilterID, Command, InpStruct, NPnt, OutStruct) ;
```

Input parameters

<i>FilterID</i> (INT)	Filter identifier
<i>Command</i> (WORD)	Command to execute
<i>InpStruct</i> (Filter_Struct)	Elements structure to filter

Output parameters

<i>NPnt</i> (INT)	Number of points still present when Command = FILTER_STOP
<i>OutStruct</i> (Filter_Struct)	Elements structure filtered

Execution mode

Immediate

Use

This function applies, re-initialize or stop (see Command table) a start or reversal filter created before the elements indicated in the *Filter_Struct* structure below.

Field	Description
ElementsNumber	Indicates the number of elements to filter
Elements	Array containing the elements to filter

Command	Description
FILTER_RUN	Runs the filter
FILTER_INIT	Calculates the new starting initial conditions of the filter
FILTER_STOP	Stops the filter

FILTER_INIT parameter reloads new starting conditions on the filter. The filter is stopped and re-initialized only if called before the FILTER_STOP command. FILTER_STOP parameter allows to stop the filter execution. The application keeps on calling the function until the *NPnt* output (showing the number of points still present) becomes =0.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	FilterID not set

Example

```
Ret : dword;
Tr : lreal := 202.0;
Tf : lreal := 202.0;
Boost : lreal := 2.2;
FilterID : int;
NPnt : int;

(* ---- Creates the filter ---- *)
Ret := Filter_Boost_Create (Tr, Tf, Boost, FilterID);

InpStruct: Filter_Struct;
OutStruct: Filter_Struct;
InpStruct.ElementsNumber := 3;
InpStruct.Elements[0] := pos;
InpStruct.Elements[1] := vel;
InpStruct.Elements[2] := acc;

(* ---- Applies the FilterID filter---- *)
Ret := Filter_Boost_Run (FilterID, FILTER_RUN, InpStruct, NPnt, OutStruct,
NPnt);
```

Applies the filter with ID1 to the pos, vel and acc elements.

See also

[Filter_Boost_Create](#), [Filter_Boost_Delete](#)

4.68 Filter_LowPass1st_Delete

The Filter_LowPass1st_Delete function deletes a first order low-pass filter.

Syntax

```
ret_code := Filter_LowPass1st_Delete (FilterID) ;
```

Input parameter

FilterID (INT) Filter identifier

Execution mode

Wait

Use

This function deletes a first order low-pass filter.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>FilterID</i> non-existent

Example

```
Ret : dword;
FilterID : int := 1;
Ret := Filter_LowPass1st_Delete (FilterID);
```

Deletes the filter with ID1.

See also

[Filter_LowPass1st_Create](#), [Filter_LowPass1st_Run](#)

4.69 Filter_Smooth_Create

The Filter_Smooth_Create creates a floating average filter.

Syntax

```
ret_code := Filter_Smooth_Create (NPnt, FilterID) ;
```

Input parameter

NPnt (INT) Number of points to calculate the average

Output parameter

FilterID (INT) Filter identifier

Execution mode

Wait

Use

This function creates a floating average filter specifying the number of input points with which calculate the average.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0005	Wrong <i>NPnt</i>
0x003A0001	<i>Maximum number of filters expired</i>

Example

```
Ret : dword;
NPnt : int := 40;
FilterID : int;
Ret := Filter_Smooth_Create (NPnt, FilterID);
```

Creates a 40 points average filter

See also

[Filter_Smooth_Delete](#), [Filter_Smooth_Run](#)

4.70 Filter_Smooth_Run

The Filter_Smooth_Run applies a floating average filter.

Syntax

```
ret_code := Filter_Smooth_Run (FilterID, Command, InpStruct, NPnt, OutStruct) ;
```

Input parameters

<i>FilterID</i> (INT)	Filter identifier
<i>Command</i> (WORD)	Command to execute
<i>InpStruct</i> (Filter_Struct)	Elements structure to filter

Output parameters

<i>NPnt</i> (INT)	Number of points still present when Command = FILTER_STOP
<i>OutStruct</i> (Filter_Struct)	Elements structure filtered

Execution mode

Immediate

Use

This function applies, re-initialize or stop (see Command table) a floating average filter created before the elements indicated in the *Filter_Struct* structure below.

Field	Description
ElementsNumber	Indicates the number of elements to filter
Elements	Array containing the elements to filter

Command	Description
FILTER_RUN	Runs the filter
FILTER_INIT	Calculates the new starting initial conditions of the filter
FILTER_STOP	Stops the filter

FILTER_INIT parameter reloads new starting conditions on the filter. The filter is stopped and re-initialized only if called before the FILTER_STOP command. FILTER_STOP parameter allows to stop the filter execution. The application keeps on calling the function until the *NPnt* output (showing the number of points still present) becomes =0.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	FilterID not set

Example

```
Ret : dword;
NPnt : int := 40;
FilterID : int;

(* ---- Creates a floating average filter ---- *)
Ret := Filter_Smooth_Create (NPnt, FilterID);

InpStruct.ElementsNumber := 2;
InpStruct.Elements[0] := pos;
InpStruct.Elements[1] := vel;

(* ---- Runs the filter ---- *)
Ret := Filter_Smooth_Run (FilterID, FILTER_RUN, InpStruct, Npnt, OutStruct);
```

Applies the filter with ID1 to pos and vel elements.

See also

[Filter_Smooth_Create](#), [Filter_Smooth_Delete](#)

4.71 Filter_Acc_Delete

The Filter_Acc_Delete function deletes an acceleration filter.

Syntax

```
ret_code := Filter_Acc_Delete (FilterID) ;
```

Input parameters

FilterID (INT) Filter identifier

Execution mode

Wait

Use

This function deletes an acceleration filter.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>FilterID</i> non-existent

Example

```
Ret : dword;
FilterID : int := 1;
Ret := Filter_Acc_Delete (FilterID);
```

Deletes the filter with ID1.

See also

[Filter_Acc_Create](#), [Filter_Acc_Run](#)

4.72 Filter_Bessel_Create

The Filter_Bessel_Create filter creates a Bessel filter.

Syntax

```
ret_code := Filter_Bessel_Create (Order, CutOffFreq, Ts, FilterID) ;
```

Input parameters

<i>Order</i> (INT)	Filter order[1..10]
<i>CutOffFreq</i> (LREAL)	Cut off frequency
<i>Ts</i> (LREAL)	Sampling time(ms)

Output parameter

<i>FilterID</i> (INT)	Filter identifier
-----------------------	-------------------

Execution mode

Wait

Use

This function creates a Bessel filter which cut off frequency and order are indicated in input.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0001	Maximum filters number expired
0x003A0004	Filter order not allowed

Example

```
Ret : dword;
Order : int := 2;
CutOffFreq : lreal := 4.0
FilterID : int;
Ts : LREAL := 10.0
Ret := Filter_Bessel_Create (Order, CutOffFreq, Ts, FilterID);
```

Create a second order Bessel filter.

See also

[Filter_Bessel_Delete](#), [Filter_Bessel_Run](#)

4.73 Filter_Bessel_Run

The Filter_Bessel_Run function applies a Bessel filter.

Syntax

```
ret_code := Filter_Bessel_Run (FilterID, Command, InpStruct, NPnt, OutStruct) ;
```

Input parameters

<i>FilterID</i> (INT)	Filter identifier
<i>Command</i> (WORD)	Command to run
<i>InpStruct</i> (Filter_Struct)	Elements structure to filter

Output parameters

<i>NPnt</i> (INT)	Number of points still present when Command = FILTER_STOP
<i>OutStruct</i> (Filter_Struct)	Elements structure filtered

Execution mode

Immediate

Use

This function applies, re-initializes or stops (see Command table) Bessel filter previously created. The element to filter is indicated in the *Filter_Struct* structure below.

Field	Description
ElementsNumber	Indicates the number of elements to filter
Elements	Array containing the elements to filter

Command	Description
FILTER_RUN	Runs the filter
FILTER_INIT	Calculates the new filter conditions
FILTER_STOP	Stops the filter

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	FilterID not set

FILTER_INIT parameter reloads new starting conditions on the filter. The filter is stopped and re-initialized only if called before the FILTER_STOP command. FILTER_STOP parameter allows to stop the filter execution. The application keeps on calling the function until the *NPnt* output (showing the number of points still present) becomes =0.

Example

```
Ret : dword;
Order : int := 2;
CutOffFreq : lreal := 4.0
FilterID : int;
NPnt : int;
Ts : LREAL := 100.0

(* ---- Creates the filter ---- *)
Ret := Filter_Bessel_Create (Order, CutOffFreq, Ts, FilterID);

InpStruct: Filter_Struct;
OutStruct: Filter_Struct;
InpStruct.ElementsNumber := 2
InpStruct.Elements[0] := pos
InpStruct.Elements[1] := vel
LastPoint : bool := false;

(* ---- Runs the FilterID filter---- *)
Ret := Filter_Bessel_Run (FilterID, FILTER_RUN, InpStruct, NPnt, OutStruct);
```

Applies the filter to pos and vel elements.

See also

[Filter_Bessel_Create](#), [Filter_Bessel_Delete](#)

4.74 Filter_Boost_Delete

The Filter_Boost_Delete function deletes a start or reversal filter.

Syntax

```
ret_code := Filter_Boost_Delete (FilterID) ;
```

Input parameters

FilterID (INT) Filter identifier

Execution mode

Wait

Use

This function deletes a start or reversal filter identified by FilterID.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>FilterID</i> non-existent

Example

```
Ret : dword;
FilterID : int := 1;
Ret := Filter_Boost_Delete (FilterID);
```

Deletes the filter with ID1.

See also

[Filter_Boost_Create](#), [Filter_Boost_Run](#)

4.75 Filter_LowPass1st_Create

The Filter_LowPass1st_Create function creates a first order low-pass filter.

Syntax

```
ret_code := Filter_LowPass1st_Create (Order, CutOffFreq, Ts, FilterID) ;
```

Input parameters

CutOffFreq (LREAL) Cut off frequency(Hz)
Ts (LREAL) Sampling time(ms)

Output parameter

FilterID (INT) Filter identifier

Execution mode

Wait

Use

This function creates a first order low-pass filter having cut off frequency specified in input.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0001	Maximum filters number expired

Example

```
Ret : dword;
CutOffFreq : lreal := 4.0
FilterID : int;
Ts : LREAL := 10.0
Ret := Filter_LowPass1st_Create (CutOffFreq, Ts, FilterID);
```

Creates a low-pass filter.

See also

[Filter_LowPass1st_Delete](#), [Filter_LowPass1st_Run](#)

4.76 Filter_LowPass1st_Run

The Filter_LowPass1st_Run function runs a first order low-pass filter.

Syntax

```
ret_code := Filter_LowPass1st_Run (FilterID, Command, InpStruct, NPnt, OutStruct) ;
```

Input parameters

<i>FilterID</i> (INT)	Filter identifier
<i>Command</i> (WORD)	Command to execute
<i>InpStruct</i> (Filter_Struct)	Elements structure to filter

Output parameters

<i>NPnt</i> (INT)	Number of points still present when Command = FILTER_STOP
<i>OutStruct</i> (Filter_Struct)	Elements structure filtered

Execution mode

Immediate

Use

This function applies, re-initialize or stop (see Command table) a first order low-pass filter created before the elements indicated in the *Filter_Struct* structure below.

Field	Description
ElementsNumber	Indicates the number of elements to filter
Elements	Array containing the elements to filter

Command	Description
FILTER_RUN	Runs the filter
FILTER_INIT	Calculates the new filter conditions
FILTER_STOP	Stops the filter

The FILTER_INIT parameter reloads new starting conditions on the filter. If called before the FILTER_STOP command, the filter stops and then is re-initialized.

The FILTER_STOP parameter stops the filter execution. The application has to keep on calling the function until the NPnt output (indicating the number of points still present) becomes zero.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	FilterID not set

Example

```
Ret : dword;
CutOffFreq : lreal := 4.0
FilterID : int;
Ts : lreal := 100.0
NPnt : int;
(* ---- Creates the filter ---- *)
Ret := Filter_LowPass1st_Create (CutOffFreq, Ts, FilterID);

InpStruct: Filter_Struct;
OutStruct: Filter_Struct;
InpStruct.ElementsNumber := 2
InpStruct.Elements[0] := pos
InpStruct.Elements[1] := vel
LastPoint : bool := false;

(* ---- Applies the filter ---- *)
Ret := Filter_LowPass1st_Run (FilterID, FILTER_RUN, InpStruct, NPnt,
OutStruct);
```

Applies the filter to pos and vel elements.

See also

[Filter_LowPass1st_Create](#), [Filter_LowPass1st_Delete](#)

4.77 Filter_Smooth_Delete

The Filter_Smooth_Delete function deletes a floating average filter.

Syntax

```
ret_code := Filter_Smooth_Delete (FilterID) ;
```

Input parameter

FilterID (INT) Filter identifier

Execution mode

Wait

Use

This function deletes an average filter.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>FilterID</i> non-existent.

Example

```
Ret : dword;  
Ret := Filter_Smooth_Delete (1);
```

Deletes filter with ID1.

See also

[Filter_Smooth_Create](#), [Filter_Smooth_Run](#)

4.78 LookUpT_Linear_Delete

The LookUpT_Linear_Delete function deletes a look-up table with linear interpolation.

Syntax

```
ret_code := LookUpT_Linear_Delete (LookUpTID) ;
```

Input parameters

LookUpTID (INT) Look-up table identifier

Execution mode

Wait

Use

This function deletes a look-up table with linear interpolation.

Return values

The following table lists all possible values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>LookUpTID</i> non-existent

Example

```
Ret : dword;
FilterID : int := 1;
Ret := LookUpT_Linear_Delete (LookUpTID) ;
```

Deletes the look-up table with ID1.

See also

[LookUpT_Linear_Create](#), [LookUpT_Linear_Run](#)

4.79 LookUpT_Spline3_Create

The LookUpT_Spline3_Create function calculates the coefficients of a cubic spline.

Syntax

```
ret_code := LookUpT_Spline3_Create(Interp, LookUpTID) ;
```

Input parameter

Interp (Interp_Struct) Data structure to interpolate

Output parameter

LookUpTID (INT) Interpolation identifier

Execution mode

Wait

Use

This function calculates the coefficients of a cubic spline starting from the *Interp_Struct* table data.

Field	Description
N	Number of points in the table
X	Vectors containing data to interpolate
Y	

Return values

The following table lists all possible values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function execute without errors
0x003A0002	Maximum number of look-up tables expired
0x003A0003	Not monotonous table - increasing / decreasing
0x003A0006	Error in calculating the spline

Example

```
Ret : dword;
Interp : Interp_Struct;
Interp.N := 30;
Interp.X := [23919.0, 24965.0, 25540.0, 25920.0, 26202.0, 27007.0, 27439.0, 27729.0, 27949.0, 28124.0, 28270.0, 28390.0, 28495.0, 28586.0, 28667.0, 28741.0, 28804.0, 29004.0, 29146.0, 29251.0, 29330.0, 29392.0, 29441.0, 29482.0, 29514.0, 29542.0];
Interp.Y :=
[0.0001087321, 0.0002086567, 0.0003086873, 0.0004087722, 0.0005086495, 0.0010085196, 0.0015085546, 0.0020085531, 0.0025084785, 0.0030086886, 0.003508827, 0.0040090624, 0.0045090104, 0.0050089187, 0.005508754, 0.006008812, 0.0065087993, 0.008507856, 0.010508007, 0.012507735, 0.014507756, 0.016508047, 0.018508079, 0.020507948, 0.022507634, 0.024508005];
```

```
LookUpTID : int;  
Ret := LookUpT_Spline3_Create(Interp, LookUpTID);
```

Calculates the coefficients of a cubic spline referred to a 30 points table.

See also

[LookUpT_Spline3_Run](#)

4.80 LookUpT_Spline3_Run

The LookUpT_Spline3_Run function runs an interpolation.

Syntax

```
ret_code := LookUpT_Spline3_Run(Interp, LookUpTID) ;
```

Input parameters

<i>X</i> (LREAL)	Element to interpolate
<i>LookUpTID</i> (INT)	Interpolation identifier

Output parameter

<i>Y</i> (LREAL)	Element in output
------------------	-------------------

Execution mode

Immediate

Use

This function calculates the Y exit dimension, interpolating the X input dimension through cubic spline using the *LookUpTIDtable* coefficients previously calculated.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>LookUpTID</i> not set

Example

```
Ret : dword;
Interp : Interp_Struc;
Interp.N := 30;
Interp.X := [23919.0, 24965.0, 25540.0, 25920.0, 26202.0, 27007.0, 27439.0, 27729.0, 27949.0, 28124.0, 28270.0, 28390.0, 28495.0, 28586.0, 28667.0, 28741.0, 28804.0, 29004.0, 29146.0, 29251.0, 29330.0, 29392.0, 29441.0, 29482.0, 29514.0, 29542.0];
Interp.Y :=
[0.0001087321, 0.0002086567, 0.0003086873, 0.0004087722,
0.0005086495, 0.0010085196, 0.0015085546, 0.0020085531,
0.0025084785, 0.0030086886, 0.003508827, 0.0040090624,
0.0045090104, 0.0050089187, 0.005508754, 0.006008812,
0.0065087993, 0.008507856, 0.010508007, 0.012507735,
0.014507756, 0.016508047, 0.018508079, 0.020507948,
0.022507634, 0.024508005];
LookUpTID : int;
Ret := LookUpT_Spline3_Create(Interp, LookUpTID);
X : LREAL := 28586.0;
Y : LREAL;
```

```
Ret := LookUpT_Spline3_Run(X, LookUpTID, Y);
```

Calculates Y value starting from X and using the Look Up Table having the LookUpTID ID.

See also

[LookUpT_Spline3_Create](#)

4.81 LookUpT_Linear_Create

The LookUpT_Linear_Create function calculates the coefficients of a linear interpolation.

Syntax

```
ret_code := LookUpT_Linear_Create (Interp, LookUpTID) ;
```

Input parameter

Interp (Interp_Struct) Data structure to interpolate

Output parameter

LookUpTID (INT) Interpolation identifier

Execution mode

Wait

Use

This function calculates the coefficients for the linear interpolation of the input structure data shown below.

Field	Description
N	Table points number
X	Vector containing data to interpolate
Y	

Return value

The following table lists all possible values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	Maximum look-up tables number expired
0x003A0003	Not monotonous table - increasing / decreasing

Example

```
Ret : dword;
Interp : Interp_Struct;
Interp.N := 30;
Interp.X := [23919.0, 24965.0, 25540.0, 25920.0, 26202.0, 27007.0, 27439.0, 27729.0, 27949.0, 28124.0, 28270.0, 28390.0, 28495.0, 28586.0, 28667.0, 28741.0, 28804.0, 29004.0, 29146.0, 29251.0, 29330.0, 29392.0, 29441.0, 29482.0, 29514.0, 29542.0];
Interp.Y :=
[0.0001087321, 0.0002086567, 0.0003086873, 0.0004087722, 0.0005086495, 0.0010085196, 0.0015085546, 0.0020085531, 0.0025084785, 0.0030086886, 0.003508827, 0.0040090624, 0.0045090104, 0.0050089187, 0.005508754, 0.006008812, 0.0065087993, 0.008507856, 0.010508007, 0.012507735, 0.014507756, 0.016508047, 0.018508079, 0.020507948, 0.022507634, 0.024508005];
LookUpTID : int;
```

```
Ret := LookUpT_Linear_Create(Interp, LookUpTID);
```

Calculates the coefficients for a linear interpolation of a 30 points table.

See also

[LookUpT_Linear_Run](#)

4.82 LookUpT_Linear_Run

The LookUpT_Linear_Run function allows to run an interpolation.

Syntax

```
ret_code := LookUpT_Linear_Run(Interp, LookUpTID) ;
```

Input parameters

<i>X</i> (LREAL)	Element to interpolate
<i>LookUpTID</i> (INT)	Interpolation identifier

Output parameter

<i>Y</i> (LREAL)	Element in output
------------------	-------------------

Execution mode

Immediate

Use

This function calculates the Y exit dimension, interpolating the X input dimension through cubic spline using the *LookUpTIDtable* coefficients previously calculated.

Return value

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>LookUpTID</i> not set

Example

```
Ret : dword;
Interp : Interp_Struc;
Interp.N := 30;
Interp.X := [23919.0, 24965.0, 25540.0, 25920.0, 26202.0, 27007.0, 27439.0, 27729.0, 27949.0, 28124.0, 28270.0, 28390.0, 28495.0, 28586.0, 28667.0, 28741.0, 28804.0, 29004.0, 29146.0, 29251.0, 29330.0, 29392.0, 29441.0, 29482.0, 29514.0, 29542.0];
Interp.Y :=
[0.0001087321, 0.0002086567, 0.0003086873, 0.0004087722, 0.0005086495, 0.0010085196, 0.0015085546, 0.0020085531, 0.0025084785, 0.0030086886, 0.003508827, 0.0040090624, 0.0045090104, 0.0050089187, 0.005508754, 0.006008812, 0.0065087993, 0.008507856, 0.010508007, 0.012507735, 0.014507756, 0.016508047, 0.018508079, 0.020507948, 0.022507634, 0.024508005];
LookUpTID : int;
Ret := LookUpT_Linear_Create(Interp, LookUpTID);
X : LREAL := 28586.0;
Y : LREAL;
```

```
Ret := LookUpT_Linear_Run(X, LookUpTID, Y);
```

Calculates Y value starting from X and using the Look Up Table having the LookUpTID ID.

See also

[LookUpT_Linear_Create](#)

4.83 LookUpT_Spline3_Delete

The LookUpT_Spline3_Delete deletes a look-up table with cubic interpolation.

Syntax

```
ret_code := LookUpT_Spline3_Delete (LookUpTID) ;
```

Input parameter

LookUpTID (INT) Look-up table identifier

Execution mode

Wait

Use

This function deletes a look-up table for a cubic spline.

Return value

The following table lists all possible values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>LookUpTID</i> non-existent

Example

```
Ret : dword;
FilterID : int := 1;
Ret := LookUpT_Spline3_Delete (LookUpTID);
```

Deletes look-up table with ID1.

See also

[LookUpT_Spline3_Create](#), [LookUpT_Spline3_Run](#)

4.84 Pid_Delete

The Pid_Delete function deletes a PID controller.

Syntax

```
ret_code := Pid_Delete (PidID) ;
```

Input parameter

PidID (INT) Controller identifier

Execution mode

Wait

Use

This function deletes a PID controller.

Return values

The following table lists all possible values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>PidID</i> non-existent.

Example

```
Ret : dword;
PidID : int := 1;
Ret := Pid_Delete (PidID);
```

Deletes a PI controller with ID 1.

See also

[Pid_Create](#), [Pid_Run](#)

4.85 Pid_Create

The Pid_Create function creates a PID controller.

Syntax

ret_code := Pid_Create (*Kp*, *Kd*, *Ki*, *Ts*, *PidID*) ;

Input parameters

<i>Kp</i> (LREAL)	Proportional gain
<i>Kd</i> (LREAL)	Derivative gain
<i>Ki</i> (LREAL)	Integral gain
<i>Ts</i> (LREAL)	Sampling time (ms)

Output parameter

<i>PidID</i> (INT)	Controller identifier
--------------------	-----------------------

Execution mode

Wait

Use

This function creates a Pid controller specifying gains and sampling time.

Return values

The following table lists all possible values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	Maximum number of Pids expired

Example

```
Ret : dword;
Kp : LREAL := 1.0;
Kd : LREAL := 0.0;
Ki : LREAL := 0.03;
Ts : LREAL := 1000.0;
PidID : int;
Ret := Pid_Create (Kp, Kd, Ki, Ts, PidID);
```

Creates a PID controller.

See also

[Pid_Delete](#), [Pid_Run](#)

4.86 Pid_Run

The Pid_Run function allows to run a PID controller.

Syntax

ret_code := Pid_Run (*PidID*, *Input*, *Output*) ;

Input parameters

<i>PidID</i> (INT)	Controller identifier
<i>Input</i> (LREAL)	Element to check

Output parameters

<i>Output</i> (LREAL)	Controller output
-----------------------	-------------------

Execution mode

Immediate

Use

This function allows to use a PID previously created to control the *Input* element.

Return values

The following table lists all possible values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003A0002	<i>PidID</i> non-existent

Example

```

Ret : dword;
Kp : LREAL := 1.0;
Kd : LREAL := 0.0;
Ki : LREAL := 0.03;
Input : LREAL;
Output : LREAL;
Ts : LREAL := 1000.0;
PidID : int;
Ret := Pid_Create (Kp, Kd, Ki, Ts, PidID);
Ret := Pid_Run(PidID, Input, Output);

```

Applies a PI controller to the Input variable.

See also

Pid_Delete, Pid_Create

5. Open_EtherCAT library: EtherCAT functions

ECAT_CommStatus	Reads the EtherCAT communication status
ECAT_ReadDiagLog	Reads the notifications of the EtherCAT communication software
ECAT_WriteDeviceData	Writes information related to a device connected on the EtherCAT bus Sercos profile
ECAT_WriteMasterInfo	Sends some information to the master of the EtherCAT communication. The information can be configuration data or command towards the fieldbus
ECAT_ReadDeviceData	Reads information related to a device connected on the EtherCAT bus Sercos profile
ECAT_GetSlaveInfo	Allows the reading of some information of an EtherCAT device in the fieldbus.
ECAT_SetSlaveInfo	Writes information on a filedbus device
SOE_ReadParameter	Reads an IDN parameter of a device with SERCOS-Over-EtherCAT profile
SOE_Command	Sends an IDN command to a device with SERCOS-Over-EtherCAT profile
SOE_WriteParameter	Writes an IDN parameter on a device with SERCOS-Over-EtherCAT profile

5.1 ECAT_CommStatus

The ECAT_CommStatus function reads the EtherCAT communication status.

Syntax

```
ret_code := ECAT_CommStatus (BoardNumber, Instance, CommStatus) ;
```

Input parameters

BoardNumber (WORD) Board number [0, 1]
Instance (WORD) Protocol instance [1, 2]

Output parameter

CommStatus (STRUCT) This structure contains information about the EtherCAT communication status

Use

The function reads information about the EtherCAT communication status. This information is grouped in the *ECAT_CommStatusStruct* structure made as follow:

FIELD	Type	Description
InitStatus	DWORD	Initialization status
ATEM_InternalError	DWORD	Reserved
CommPhase	DWORD	Communication phase
NumConfiguredSlave	DWORD	Number of nodes configured
NumActualSlave	DWORD	Number of nodes not found on the bus
NumLogDiags	DWORD	Number of diagnostic messages

For the *InitStatus* field encoding refer to the following table:

Value	Name	Meaning
0	ECAT_MSTS_MASTER_NOT_INIT	Communication SW not initialized
1	ECAT_MSTS_MASTER_INITILZD	Communication SW initialized
2	ECAT_MSTS_MASTER_CONFIGRD	Communication SW set
3	ECAT_MSTS_MASTER_BUSSCAND	Bus scan complete

The *InitStatus* field reports the encoding phases of the communication software; with no field errors it returns value 3, which indicates the successful bus scan and that all the devices set were found on the bus.

The *CommPhase* field indicates the bus communication phase, please refer to the following table:

Value	Name	Meaning
0	ECAT_CP_NODEF	Bus not communicating
1	ECAT_CP_INIT	Bus INIT phase
2	ECAT_CP_PREOP	Bus PREOP phase
3	ECAT_CP_SAFEOP	Bus SAFEOP phase
4	ECAT_CP_OP	Bus OP phase

Work stage 4 is reached if there are no errors.

The *NumLogDiags* field indicates the number of notifications generated by the communication software. If there are no errors, value is always 0. When value is higher than 0, there is at least one communication error which cause can be read using the “ECAT_ReadDiagLog” function.

Return values

Value (HEX)	Description
0	Right execution of the function
0x00360002	Parameter out of range
0x00360003	Protocol not instantiated
0x00360016	Invalid board

Example

```
VAR
Ret_code:  DWORD ;
Status : ECAT_CommStatusStruct;

END_VAR
(* Reads board 1 communication status, first instance *)
Ret_code := ECAT_CommStatus (1, 1, Status);
```

See also

[ECAT_ReadDiagLog](#)

5.2 ECAT_ReadDiagLog

The ECAT_ReadDiagLog function reads the notifications of the EtherCAT communication software.

Syntax

```
ret_code := ECAT_ReadDiagLog (BoardNumber, Instance, DiagLogEntry);
```

Input parameters

BoardNumber (WORD) Board number [0, 1]
Instance (WORD) Protocol instance [1, 2]

Output parameters

DiagLogEntry (STRUCT) This structure contains diagnostic information

Use

The function reads the diagnostic notifications of the EtherCAT communication master. The function ECAT_CommStatus reports if there are any notifications.

Information is grouped in the ECAT_DiagLogEntryStruct structure made as follow:

FIELD	Type	Description
NotifyCode	DWORD	Notification code
StationAddress	DWORD	Node address (if available)
Param1	DWORD	Parameter1 (depending on the notification)
Param2	DWORD	Parameter 2 (depending on the notification)
Param3	DWORD	Parameter 3 (depending on the notification)
Param4	DWORD	Parameter 4 (depending on the notification)
Param5	DWORD	Parameter 5 (depending on the notification)
Param6	DWORD	Parameter 6 (depending on the notification)

Please, refer to the following table for encoding the *NotifyCode* field:

Value	Meaning
0x00010001	Working Counter error
0x00010002	Master Init Working Counter error
0x00010003	Slave Init Working Counter error
0x00010004	EOE MBox error RX Working Counter
0x00010005	COE MBox error RX Working Counter
0x00010006	FOE MBox error RX Working Counter
0x00010007	EOE MBox error TX Working Counter
0x00010008	COE MBox error TX Working Counter
0x00010009	FOE MBox error TX Working Counter
0x0001000A	No feedback after sending Ethernet frame
0x0001000B	No slave feedback after INIT command sending
0x0001000C	No feedback after INIT command sending
0x0001000D	No feedback command after Ethernet frame sending
0x0001000E	MBox timeout on command INIT
0x0001000F	Not all slaves in operative mode

0x00010010	Ethernet cable disconnected
0x00010011	Reserved
0x00010012	Redundancy check: Ethernet cable interrupted
0x00010013	Slave in error status
0x00010014	Slave information in error
0x00010015	Slave missing or fix address lost
0x00010016	SOE Mbox error RX Working Counter
0x00010017	SOE Mbox error TX Working Counter
0x00010018	SOE Mbox feedback with error
0x00010019	COE Mbox SDO abort
0x0001001A	Client registering error
0x0001001B	Redundancy check: Ethernet cable restored
0x0001001C	FOE Mbox abort
0x0001001D	Invalid data from Mbox
0x0001001E	PDI watchdog expired on the slave
0x0001001F	Slave not supported
0x00010020	Slave in not consistent status
0x00010021	Reserved
0x00010022	VOE Mbox error TX Working Counter
0x00010023	EEPROM checksum incorrect

Return values

Value(HEX)	Description
0	Right execution of the function
0x00360002	Parameter out of range
0x00360003	Protocol non-initialized
0x00360014	No notifications
0x00360015	Some notifications are missing but reading can continue
0x00360016	Invalid board

Example

```

VAR
  Ret_code:  DWORD ;
  DiagLogEntry: ECAT_DiagLogEntryStruct;

END_VAR

Ret_code := ECAT_ReadDiagLog (1, 1, DiagLogEntry);

```

See also

[ECAT_CommStatus](#)

5.3 ECAT_WriteDeviceData

The ECAT_WriteDeviceData function writes information related to a device connected on the EtherCAT bus Sercos profile.

Syntax

```
ret_code := ECAT_WriteDeviceData (BoardNumber, NodeAddress, InfoCode, Value);
```

Input parameters

<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Node address on the bus [1,128]
<i>InfoCode</i> (WORD)	Information code to write
<i>Value</i> (ANY_ELEMENTARY)	Value to write. This information can be BYTE, WORD or DWORD type.

Use

Function allows to write following information:

<i>InfoCode</i>	Description	Format
1	Device Control Word	WORD
3	Connection Control Word	WORD



The function should be called at 0 priority within a task.

Return values

<i>Value</i> (HEX)	Description
0	Right execution of the function
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00340006	<i>InfoCode</i> parameter out of range

Example

```
VAR
Ret_code: DWORD;
NodeAddress : WORD := 7;
InfoCode: WORD := 1; (* Write control word *)
DeviceData: DWORD;

END_VAR

Ret_code := ECAT_WriteDeviceData (0, NodeAddress, InfoCode, DeviceData);
```

See also

[ECAT_ReadDeviceData](#)

5.4 ECAT_WriteMasterInfo

The function ECAT_WriteMasterInfo sends some information to the master of the EtherCAT communication. The information can be configuration data or command towards the fieldbus.

Syntax

```
ret_code := ECAT_WriteMasterInfo (BoardNumber, Instance, ParamID, MasterInfo) ;
```

Inout parameters

<i>BoardNumber</i> (WORD)	Board number [0, 1]
<i>Instance</i> (WORD)	Protocol instance [1, 2]
<i>ParamID</i> (WORD)	Parameter/function identifier
<i>MasterInfo</i> (ARRAY[0..7] OF DWORD)	Input sub-parameters

Output parameters

<i>MasterInfo</i> (ARRAY[0..7] OF DWORD)	Output sub-parameters
--	-----------------------

Execution mode

Immediate or Wait according to the parameter to write/function to enable.

Use

The function can be used to access the master of the EtherCAT filedbus communication for an advanced management and for diagnostics.

ParamID	Mnemonic name	Modality	Description
0	-	-	Reserved
1	ECAT_MSWPAR_COMMMPHA	Immediate	Sets the communication pahse for the whole bus

Parameter ECAT_MSWPAR_COMMMPHA

Sets the communication pahes for th whole fieldbus. The Sub-parameters to shift in input are:

Sub-parameter	Description
MasterInfo[0]	Sets the communication phase of all the bus. The accepted values are: 1=INIT, 2=PREOP, 3= SAFEOP, 4=OP. Use the function ECAT_CommStatus to read the present communication phase.

The function has an immediate return, but the realtransiton on the filedbus requires several data exchange cycles. To know the present status of the communication, use the function ECAT_CommStatus.

There are no output Sub-parameters.

Return values

Value (HEX)	Description
0x00000000	Function executed without errors
0x00360002	Out of range parameter
0x00360003	Protocol not istantiated
0x00360016	Invalid board
0x9811nnnn	Errors of the communication master (see application manual)

Example

```
Ret      : dword;
Board    : word := 0;
ParamID  : word := 1;
MasterInfo : ARRAY[0..7] of dword;

MasterInfo[0] := 1;
Ret := ECAT_WriteMasterInfo (Board, 1, ParamID, MasterInfo);
```

Sets the INIT communication stage.

See also

[ECAT_CommStatus](#)

5.5 ECAT_ReadDeviceData

The ECAT_ReadDeviceData function reads information related to a device connected on the EtherCAT bus Sercos profile.

Syntax

```
ret_code := ECAT_ReadDeviceData (BoardNumber, NodeAddress, InfoCode, Value) ;
```

Input parameters

<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Node address on the bus [1,128]
<i>InfoCode</i> (WORD)	Information code to read

Output parameter

<i>Value</i> (ANY_ELEMENTARY)	Read value. This information can be BYTE, WORD or DWORD type.
-------------------------------	---

Use

The function allow to read the following information:

<i>InfoCode</i>	Description	Format
0	Device Status Word	WORD
1	Device Control Word	WORD
3	Connection Control Word	WORD

Return values

<i>Value</i> (HEX)	Description
0	Right execution of the function
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00340006	<i>InfoCode</i> parameter out of range

Example

```
VAR
  Ret_code: DWORD;
  NodeAddress : WORD := 7;
  InfoCode: WORD := 0; (* Read status word *)
  DeviceData: WORD;

  END_VAR

  Ret_code := ECAT_ReadDeviceData (0, NodeAddress, InfoCode, DeviceData);
```

See also

[ECAT_WriteDeviceData](#)

5.6 ECAT_GetSlaveInfo

The function ECAT_GetSlaveInfo reads of some information of an EtherCAT device in the fieldbus.

Syntax

```
ret_code := ECAT_GetSlaveInfo (BoardNumber, Instance, NodeIndex, ParamID, SlaveInfo);
```

Input parameters

<i>BoardNumber</i> (WORD)	Board number [0, 1]
<i>Instance</i> (WORD)	Protocol instance [1, 2]
<i>NodeIndex</i> (WORD)	Slave topological index [0,1,2...4095]
<i>ParamID</i> (WORD)	Parameter identifier
<i>SlaveInfo</i> (ARRAY[0..19] OF DWORD)	Input sub-parameters

Output parameters

<i>SlaveInfo</i> (ARRAY[0..19] OF DWORD)	Output sub-parameters
--	-----------------------

Use

The function can be used to acces a remote device of the EtherCAT fieldbus for an andvance management and for diagnostics.

ParamID	Mnemonic name	Modality	Description
0	-	-	Reserved
1	-	-	Reserved
2	-	-	Reserved
3	ECAT_GETSL_COMMSTS	Wait	Reads the communication stage
4	ECAT_GETSL_BASEINF	Wait	Reads basic information
5	ECAT_GETSL_STATUS1	Wait	Reads status information
6	-	-	Reserved

ECAT_GETSL_COMMSTS parameter

Reads the communication stage of the device.

The sub-parameters in input are:

Sub-parameter	Description
SlaveInfo[0]	Timeout (msec) for the execution (indicate a value higher than 0, for example 100)

The sub-parameters in output are:

Sub-parameter	Description
SlaveInfo[1]	Current status of the communication
SlaveInfo[2]	Required status of the communication

The values can be

Communication status (HEX)	Description
0x0000000001	INIT communication status
0x0000000002	PREOP communication status
0x0000000004	SAFEOP communication status
0x0000000008	OP communication status
Altri valori	The node is in error or cannot be reached

ECAT_GETSL_BASEINF parameter

Reads some basic information

The output sub-parameters are:

Sub-parameter	Description
SlaveInfo[0]	low word: Station Address, high word Alias
SlaveInfo[1]	Vendor ID
SlaveInfo[2]	Product Code
SlaveInfo[3]	Revision Number
SlaveInfo[4]	Serial Number
SlaveInfo[5..19]	Nome ASCII

ECAT_GETSL_STATUS1 parameter

Legge alcune informazioni riguardo lo stato del dispositivo.

The output sub-parameters are:

Sub-parameter	Description
SlaveInfo[0]	Bit 0 : present (yes / no) Bit 1 : opzionale (yes / no)
SlaveInfo[1]	Error code
SlaveInfo[2]	Error code of the bus scanning
SlaveInfo[3]	low word: AL register status , high word: AL register code
SlaveInfo[4]	low word: ports status, high word: ports type

AL register status (bit)	Description
3..0	Communication status (see previous table)
4	Error flag
15..5	Reserved

AL register status (HEX)	Description
0x0000	No error
0x0001	Unspecified error
0x0002	No memory
0x0011	Invalid requested state change
0x0012	Unknown requested state
0x0013	Bootstrap not supported
0x0014	No valid firmware
0x0015	Invalid mailbox configuration
0x0016	Invalid mailbox configuration

0x0017	Invalid sync manager configuration
0x0018	No valid inputs available
0x0019	No valid outputs
0x001A	Synchronization error
0x001B	Sync manager watchdog
0x001C	Invalid Sync Manager Types
0x001D	Invalid Output Configuration
0x001E	Invalid Input Configuration
0x001F	Invalid Watchdog Configuration
0x0020	Slave needs cold start
0x0021	Slave needs INIT
0x0022	Slave needs PREOP
0x0023	Slave needs SAFEOP
0x0024	Invalid input mapping
0x0025	Invalid output mapping
0x0026	Inconsistent settings
0x0027	Free-Run not supported
0x0028	Synchronization not supported
0x0029	Free-Run needs 3 buffer mode
0x002A	Background watchdog
0x002B	No valid inputs and outputs
0x002C	Fatal Sync error
0x002D	No Sync error
0x0030	Invalid DC SYNCH Configuration
0x0031	Invalid DC Latch Configuration
0x0032	PLL Error
0x0033	Invalid DC IO Error
0x0034	Invalid DC Timeout Error
0x0035	DC invalid Sync Cycle Time
0x0036	DC Sync0 Cycle Time
0x0037	DC Sync1 Cycle Time
0x0041	MBX_AOE
0x0042	MBX_EOE
0x0043	MBX_COE
0x0044	MBX_FOE
0x0045	MBX_SOE
0x004F	MBX_VOE
0x0050	EEPROM no access
0x0051	EEPROM error
0x0060	Slave restarted locally
Other codes < 0x8000	Reserved
0x8000-0xFFFF	Vendor specific

Ports status	Description
3..0	Device connected yes / no (each bit corresponds to a port)
4..7	Connection established yes / no (each bit corresponds to a port)
8..11	Port closed yes / no (each bit corresponds to a port)
12..15	Signal detected yes / no (each bit corresponds to a port)

Ports type	Description
1..0	Port 0 (00=not present, 01=not configured, 10=EBUS, 11=MII/RMII)
3..2	Port 1 (as above)
5..4	Port 2 (as above)
7..6	Port 3 (as above)
15..8	N/A

Return values

Value (HEX)	Description
0x00000000	Function executed without errors
0x00360002	Out of range parameter
0x00360003	Protocol not instanciated
0x00360016	Invalid board
0x9811nnnn	Errors of the communication master (see application manual)

Example

```

Ret      : dword;
Board     : word := 0;
ParamID   : word := 4;
NodeIndex : word := 0;
SlaveInfo : array[0..19] of dword;
Ret := ECAT_GetSlaveInfo (Board, 1, NodeIndex, ParamID, MasterInfo);

```

Reads the basic information of the first device in the line.

See also

[ECAT_SetSlaveInfo](#)

5.7 ECAT_SetSlaveInfo

The function ECAT_SetSlaveInfo writes some information on an EtherCAT device on the fieldbus. The information can be configuration data or commands towards the device.

Syntax

```
ret_code := ECAT_SetSlaveInfo (BoardNumber, Instance, NodeIndex, ParamID, SlaveInfo);
```

Input parameters

<i>BoardNumber</i> (WORD)	Board number [0, 1]
<i>Instance</i> (WORD)	Protocol instance [1, 2]
<i>NodeIndex</i> (WORD)	Slave topological index [0,1,2...4095]
<i>ParamID</i> (WORD)	Parameter identifier
<i>SlaveInfo</i> (ARRAY[0..19] OF DWORD)	Input sub-parameters

Output parameters

SlaveInfo (ARRAY[0..19] OF DWORD) Output sub-parameters

Execution mode

Immediate or Wait according to the parameter to be written or according to the function to enable.

Use

The function can be used for writing configuration data or enabling functionalities on a remote device of the EtherCAT fieldbus for advanced management and diagnostics.

ParamID	Mnemonic name	Modality	Description
3	ECAT_SETSL_COMMMPHA	Wait	Sets the communication phase

ECAT_SETSL_COMMMPHA parameter

Sets the communication phase of the device while the master is in operative status (OP).

The Sub-parameters to switch as inputs are:

Sub-parameter	Description
SlaveInfo[0]	Communication stage to reach
SlaveInfo[1]	Timeout (msec) for the execution (indicates a value higher than 0, for example 100)

The values that can be used are:

Fase di Comunicazione (HEX)	Description
0x00000001	INIT communication stage
0x00000002	PREOP communication stage
0x00000004	SAFEOP communication stage
0x00000008	OP communication stage

In order to know the current status of the device, the function ECAT_GetSlaveInfo can be used. There are no output sub-parameters.

Return values

The following table lists the values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00360002	Out of range parameter
0x00360003	Non-instanced protocol
0x00360016	Invalid board
0x9811nnnn	Errors of the communication master (see application manual)

Example

```

Ret      : dword;
Board    : word := 0;
ParamID  : word := 3;
NodeIndex : word := 0;
SlaveInfo : array[0..19] of dword;

SlaveInfo [0] := 2;
SlaveInfo [1] := 100;
Ret := ECAT_SetSlaveInfo (Board, 1, NodeIndex, ParamID, MasterInfo);

```

Sets the PREOP communication pahse for the first device in the line.

See also

[ECAT_GetSlaveInfo](#)

5.8 SOE_ReadParameter

The SOE_ReadParameter function reads an IDN parameter of a device connected on EtherCAT bus SERCOS I profile.

Syntax

```
ret_code := SOE_ReadParameter (ExecMode, ExecStatus, BoardNumber, NodeAddress,  
                          ModuleIndex, IDN, IDNelement, ValuesNum, Values, Timeout);
```

Input parameters

<i>ExecMode</i> (INT)	Execution mode [MODE_WAIT, MODE_POLLING]
<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Bus node address [1..128]
<i>ModuleIndex</i> (WORD)	Modulus index
<i>IDN</i> (DWORD)	IDN parameter to read
<i>IDNelement</i> (INT)	IDN element
<i>Timeout</i> (DINT)	Maximum execution time

Output parameters

<i>ExecStatus</i> (DWORD)	Execution status
<i>ValuesNum</i> (INT)	Number of values read by the function
<i>Values</i> (ANY)	Read value. This information can be BYTE, WORD, DWORD ARRAY OF BYTE, ARRAY OF WORD, ARRAY OF DWORD and STRING type.

Use

For the address device *NodeAddress*, function Reads the information shown in *IDNelement* of an *IDN* parameter; the values can be:

<i>IDNelement</i>	Description	Format
1	IDN status	WORD
2	Nome IDN name	STRING
3	Attribute	DWORD
4	Unit of measure	STRING
5	Minimum value	(*)
6	Maximum value	(*)
7	Current parameter value	(*)

(*) format depends on the parameter type



Current parameter value is expressed in drive internal units; the attribute value should be read and interpreted to be converted in its unit of measure (element 3 of the IDN).

Attribute decoding the following table:

Bits	Description
31	Reserved
30	=1 Write protection in CP4
29	=1 Write protection in CP3
28	=1 Write protection in CP2
27..24	Decimal point position by input and visualization (can't be applied to the floating point)
23	Reserved
22..20	Data type coding and format viewing: 000 - Binary 001 - Decimal integer not signed 010 - Decimal integer signed 011 - Hexadecimal integer not signed 100 - Text 101 - IDN 110 - ANSI 754-1985 floating point number 111 - Reserved
19	Function (0=data/parameter, 1=command)
18..16	Data length: 000 - Reserved 001 - 2 bytes 010 - 4 bytes 011 - 8 bytes 100 - variable length / 1 byte record 101 - variable length / 2 bytes record 110 - variable length / 4 bytes record 111 - variable length / 8 bytes record
15..0	Conversion factor for input/output and visualization

In MODE_WAIT mode, function waits for the parameter to be read for the maximum time indicated in the *Timeout* parameter (expressed in milliseconds). In order to avoid the premature exit of the function while the parameter reading is still running, it can be advisable to indicate a *Timeout* value long enough (for instance, some seconds).

In MODE_POLLING mode, the function has to be recalled until the *ExecStatus* parameter is equal to COMMAND_RUNNING. In this mode the user manages the timeout management.

The *ValuesNum* output parameter has the following values:

- ▷ value 1 if the *Values* parameter is scalar type (that is BYTE, WORD, DWORD)
- ▷ number of elements read if the *Values* parameter is array type
- ▷ string length if the *Values* parameter is STRING type

Return values

Value (HEX)	Description
0	Right function execution
0x00338003	Invalid execution mode, check <i>ExecMode</i>
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00338031	Busy node
0x00338032	Node in not recoverable communication error
0x00338034	Dwell time for communication request has expired
0x00338042	Invalid <i>IDNelement</i>
0x00338043	Data cut (data dimension too big)

See also Service Channel internal errors table

Example

```

VAR
  Ret_code: DWORD;
  ExecMode : INT := MODE_WAIT;
  ExecStatus : DWORD;
  NodeAddress : WORD := 2;
  IDN : DWORD := 51; (* Position feedback value 1 *)
  IDNelement : INT := 7; (* Read current value *)
  ValuesNum : INT;
  Value : DWORD;
  Timeout: DINT := 10000; (* 10 sec *)

END_VAR
  Ret_code := SOE_ReadParameter(ExecMode, ExecStatus, 0, NodeAddress,
                                IDN, IDNelement, ValuesNum, Value,Timeout);

```

See also

[SOE_WriteParameter](#)

5.9 SOE_Command

The SOE_Command function runs an IDN command of a device connected on an EtherCAT bus with Sercos profile.

Syntax

```
ret_code := SOE_Command (ExecMode, ExecStatus, BoardNumber, NodeAddress, ModulusIndex,  
                  IDNcommand, Timeout);
```

Input parameters

<i>ExecMode</i> (INT)	Execution mode [MODE_WAIT, MODE_POLLING]
<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Bus node address [1..128]
<i>ModuleIndex</i> (WORD)	Module index
<i>IDNcommand</i> (DWORD)	IDN command to execute
<i>Timeout</i> (DINT)	Maximum execution time

Output parameter

<i>ExecStatus</i> (DWORD)	Execution status
---------------------------	------------------

Use

The function sends the *IDNcommand* IDN command to the device with *NodeAddress* address.

In MODE_WAIT mode, the function waits for the command to finish for the maximum time indicated by the *Timeout* parameter (expressed in milliseconds). It is advisable to indicate a timeout value long enough (for instance some seconds, according to the command type) to avoid the early output of the function while the command on the device is still running.

In MODE_POLLING mode the function has to be recalled until the *ExecStatus* parameter is equal to COMMAND_RUNNING. In this mode the timeout is managed by the user .

Return values

Value (HEX)	Description
0	Function executed without errors
0x00338003	Invalid execution mode, check <i>ExecMode</i>
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00338031	Busy node
0x00338032	Node communication errors not recoverable
0x00338034	Communication request dwell time expired
0x00338035	Command negative acknowledge

See also the Internal errors returned by Service Channel table.

Example

VAR

```
Ret_code:  DWORD;
ExecMode : INT = MODE_WAIT;
ExecStatus : DWORD;
NodeAddress : WORD := 7;
IDNcommand : DWORD := 99; (* Reset alarm *)
Timeout: DINT:= 10000; (* 10 sec *)
```

END_VAR

```
Ret_code := SOE_Command (ExecMode, ExecStatus, 0, NodeAddress,
                         IDNcommand, Timeout);
```

5.10 SOE_WriteParameter

The SOE_Command function allows to write and IDN command of a device connected on an EtherCAT bus with Sercos profile.

Syntax

```
ret_code := SOE_WriteParameter (ExecMode, ExecStatus, BoardNumber, NodeAddress,  
                          ModuleIndex, IDN, IDNelement, ValuesNum, Values, Timeout);
```

Output parameters

<i>ExecMode</i> (INT)	Execution mode [MODE_WAIT, MODE_POLLING]
<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Node address on the bus [1..128]
<i>ModuleIndex</i> (WORD)	Modulus index
<i>IDN</i> (DWORD)	IDN parameter to write
<i>IDNelement</i> (INT)	IDN element
<i>ValuesNum</i> (INT)	Number of values to write
<i>Values</i> (ANY)	Value to write. This parameter can be BYTE, WORD, DWORD, ARRAY OF BYTE, ARRAY OF WORD, ARRAY OF DWORD, STRING type
<i>Timeout</i> (DINT)	Maximum execution time

Output parameter

<i>ExecStatus</i> (DWORD)	Execution status
---------------------------	------------------

Use

For the *NodeAddress* address device, the function writes information of the IDN parameter in *IDNelement* (see S3_ReadParameter function). Usually, the information to write is the current parameter value, therefore set *IDNelement*=7.



The current parameter value is expressed in units internal to the drive; for the conversion on its internal unit, read and process the attribute value (see S3_ReadParameter function for the coding).

In MODE_WAIT mode, the function waits for the command to finish for the maximum time indicated by the *Timeout* parameter (expressed in milliseconds). It is advisable to indicate a timeout value long enough (for instance some seconds, according to the command type) to avoid the early output of the function while the command on the device is still running.

In MODE_POLLING mode the function has to be recalled until the *ExecStatus* parameter is equal to COMMAND_RUNNING. In this mode the timeout is managed by the user .

ValuesNum input parameter must have these values:

- ▷ If *Values* parameter is scaled (that is BYTE, WORD, DWORD) value is 1
- ▷ If *Values* parameter is an array type, the number of elements to write

Return values

Value (HEX)	Description
0	Function executed without errors
0x00338003	Invalid execution mode, check <i>ExecMode</i>
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00338031	Node busy
0x00338032	Node communication errors not recoverable
0x00338034	Communication request dwell time expired
0x00338042	Invalid <i>IDNelement</i>
0x00338043	Data cut data dimension is too big)

See also the Internal errors returned by **Service Channel table**.

Example

```

VAR
  Ret_code:  DWORD;
  ExecMode : INT := MODE_WAIT;
  ExecStatus : DWORD;
  NodeAddress : WORD := 2;
  IDN : DWORD := 104; (* Position loop Kv value *)
  IDNelement : INT:= 7; (* Write current value *)
  Value : WORD := 3; (* Internal unit *)
  Timeout: DINT := 10000; (* 10 sec *)

END_VAR
Ret_code := SOE_WriteParameter (ExecMode, ExecStatus, 0, NodeAddress,
                                IDN, IDNelement, 1, Value, Timeout);

```

See also

[ECAT_ReadParameter](#)

5.11 Service Channel internal errors table

0x00330001	Service Channel not open
0x00330009	Invalid access when closing the Service Channel
0x00331001	Invalid IDN
0x00331009	Invalid access to 1
0x00332001	IDN name missing
0x00332002	IDN name transmission too short
0x00332003	IDN name transmission too long
0x00332004	IDN name cannot be changed (read only)
0x00332005	IDN name cannot be written at the moment
0x00333002	IDN attribute transmission too short
0x00333003	IDN attribute transmission too long
0x00333004	Attribute cannot be changed (read only)
0x00333005	The attribute cannot be written at the moment
0x00334001	Unit of measure missing
0x00334002	Measure of unit transmission too short
0x00334003	Measure of unit transmission too long
0x00334004	The measure of unit cannot be changed (read only)
0x00334005	The measure of unit cannot be written at the moment
0x00335001	Minimum value missing
0x00335002	Minimum value transmission too short
0x00335003	Minimum value transmission too long
0x00335004	Minimum value cannot be changed (read only)
0x00335005	Minimum value cannot be written at the moment
0x00336001	Minimum value missing
0x00336002	Maximum value transmission too short
0x00336003	Maximum value transmission too long
0x00336004	Maximum value cannot be changed (read only)
0x00336005	The maximum value cannot be written at the moment
0x00337002	Value transmission too short
0x00337003	Value transmission too long
0x00337004	Read only value
0x00337005	The value cannot be written at this communication stage
0x00337006	Value is lower than the minimum allowed
0x00337007	Value exceeds the maximum allowed
0x00337008	The value of the number or the bit combination is invalid
0x00337009	The values is write protected by a password
0x0033700A	The value is write protected
0x0033700B	Invalid indirect addressing (for IDN list)
0x0033700C	The value is write protected because of other settings
0x0033700D	The value expressed in floating point is invalid
0x0033700E	The value is write protected at parameter level
0x0033700F	The value is write protected at operative level
0x00337010	Command already active
0x00337011	Command cannot be stopped
0x00337012	Command cannot be executed at the moment

0x00337013	Invalid parameters: command cannot be executed
------------	--

6. Open_SERCOSIII library: SERCOSIII functions

S3_Command	Runs an IDN command of a device connected on SERCOSIII bus
S3_ReadDeviceData	Reads information referred to a device connected on a device with SERCOSIII bus
S3_ReadParameter	Reads an IDN parameter of a device connected on SERCOSIII bus
S3_WriteDeviceData	Writes information referred to a device connected on a device with SERCOSIII bus
S3_CommStatus	Reads the SERCOSIII communication status
S3_ReadDiagLog	Reads the diagnostics internal to the communication master
S3_WriteParameter	Writes an IDN parameter of a device connected on SERCOSIII bus

6.1 S3_Command

The S3_Command function runs an IDN command of a device connected on **SERCOSIII** bus.

Syntax

```
ret_code := S3_Command (ExecMode, ExecStatus, BoardNumber, NodeAddress, ModulusIndex,  
                  IDNcommand, Timeout);
```

Input parameters

<i>ExecMode</i> (INT)	Execution mode[MODE_WAIT, MODE_POLLING]
<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Node address on the bus [1..128]
<i>ModuleIndex</i> (WORD)	Modulus index
<i>IDNcommand</i> (DWORD)	IDN command to run
<i>Timeout</i> (DINT)	Maximum execution time

Output parameter

<i>ExecStatus</i> (DWORD)	Execution status
---------------------------	------------------

Use

The function sends the *IDNcommand* IDN command to the device with *NodeAddress* address.

In MODE_WAIT mode, the function waits for the command to finish for the maximum time indicated by the *Timeout* parameter (expressed in milliseconds). It is advisable to indicate a timeout value long enough (for instance some seconds, according to the command type) to avoid the early output of the function while the command on the device is still running.

In MODE_POLLING mode the function has to be recalled until the *ExecStatus* parameter is equal to COMMAND_RUNNING. In this mode the timeout is managed by the user .

Return values

Value (HEX)	Description
0	Function executed without errors
0x00338003	Invalid execution mode, check ExecMode
0x00338004	Invalid board, check BoardNumber
0x00338030	Node not found, check NodeAddress
0x00338031	Node busy
0x00338032	Node communication errors not recoverable
0x00338034	Communication request dwell time expired
0x00338035	Negative acknowledge of the command

See also the Internal errors returned by Service Channel table.

Example

```
VAR
  Ret_code:  DWORD;
  ExecMode : INT = MODE_WAIT;
  ExecStatus : DWORD;
  NodeAddress : WORD := 7;
  IDNcommand : DWORD := 99; (* Reset alarm *)
  Timeout: DINT:= 10000; (* 10 sec *)

END_VAR

Ret_code := S3_Command (ExecMode, ExecStatus, 0, NodeAddress, IDNcommand,
Timeout);
```

6.2 S3_ReadDeviceData

The S3_ReadDeviceData function reads information referred to a device connected on a device with SERCOSIII bus.

Syntax

```
ret_code := S3_ReadDeviceData (BoardNumber, NodeAddress, InfoCode, Value) ;
```

Input parameters

<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Node address on the bus [1,128]
<i>InfoCode</i> (WORD)	Information code to read

Output parameter

<i>Value</i> (ANY_ELEMENTARY)	Read value. This information can be BYTE, WORD or DWORD type.
-------------------------------	---

Use

The function reads the information below:

InfoCode	Description	Format
0	Device Status Word	WORD
1	Device Control Word	WORD
3	Connection Control Word	WORD

Return values

Value (HEX)	Description
0	Function executed without errors
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00340006	<i>InfoCode</i> parameter out of range

Example

```
VAR
  Ret_code: DWORD;
  NodeAddress : WORD := 7;
  InfoCode: WORD := 0; (* Read status word *)
  DeviceData: WORD;

  END_VAR

  Ret_code := S3_ReadDeviceData (0, NodeAddress, InfoCode, DeviceData);
```

See also

[S3_WriteDeviceData](#)

6.3 S3_ReadParameter

The S3_ReadParameter reads an IDN parameter of a device connected on SERCOSIII bus.

Syntax

```
ret_code := S3_ReadParameter (ExecMode, ExecStatus, BoardNumber, NodeAddress,
                             ModuleIndex, IDN, IDNelement, ValuesNum, Values, Timeout);
```

Input parameters

<i>ExecMode</i> (INT)	Execution mode [MODE_WAIT, MODE_POLLING]
<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Node address on the bus [1..128]
<i>ModuleIndex</i> (WORD)	Modulus index
<i>IDN</i> (DWORD)	IDN parameter to read
<i>IDNelement</i> (INT)	IDN element
<i>Timeout</i> (DINT)	Maximum execution time

Output parameters

<i>ExecStatus</i> (DWORD)	Execution status
<i>ValuesNum</i> (INT)	Number of values read by the function
<i>Values</i> (ANY)	Read values. This parameter can be BYTE, WORD, DWORD, ARRAY OF BYTE, ARRAY OF WORD, ARRAY OF DWORD, STRING type

Use

For the *NodeAddress* address device, function reads the information shown in *IDNelement* of an IDN parameter; this can be:

<i>IDNelement</i>	Description	Format
1	IDN status	WORD
2	IDN name	STRING
3	Attribute	DWORD
4	Unit of measure	STRING
5	Minimum value	(*)
6	Maximum value	(*)
7	Current parameter value	(*)

(*) format depending on the parameter type



The current parameter value is expressed in units internal to the drive; for the conversion on its measure unit, read and process the attribute value (IDN element 3).

Attribute codification the following table:

Bits	Description
31	Reserved
30	=1 write protection in CP4
29	=1 write protection in CP3
28	=1 write protection in CP2
27..24	Decimal point position for input e displaying (not to be applied on the floating point)
23	Reserved
22..20	Codification according to data type and display format: 000 - Binary 001 - Decimal integer not signed 010 - Decimal integer signed 011 - Hexadecimal integer not signed 100 - Text 101 - IDN 110 - ANSI 754-1985 floating point number 111 - Reserved
19	Function (0=data/parameter, 1=command)
18..16	Data length: 000 - Reserved 001 - 2 bytes 010 - 4 bytes 011 - 8 bytes 100 - variable length / 1 byte record 101 - variable length / 2 bytes record 110 - variable length / 4 bytes record 111 - variable length / 8 bytes record
15..0	Conversion factor for input/output and displaying

In MODE_WAIT mode, the function waits for the command to finish for the maximum time indicated by the *Timeout* parameter (expressed in milliseconds). It is advisable to indicate a timeout value long enough (for instance some seconds, according to the command type) to avoid the early output of the function while the command on the device is still running.

In MODE_POLLING mode the function has to be recalled until the *ExecStatus* parameter is equal to COMMAND_RUNNING. In this mode the timeout is managed by the user .

ValuesNum output parameter can have following values:

- ▷ If *Values* parameter is scaled type (that is BYTE, WORD, DWORD) value is 1
- ▷ the *Values* parameter is array type, the number of elements read
- ▷ the *Values* parameter is STRING type, the string length

Return values

Value (HEX)	Description
0	Function executed without errors
0x00338003	Invalid execution mode, check <i>ExecMode</i>
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00338031	Node busy
0x00338032	Node communication errors not recoverable
0x00338034	Communication request dwell time expired
0x00338035	Negative acknowledge of the command
0x00338042	Invalid <i>IDNelement</i>
0x00338043	Data cut (data dimension is too big)

See also the [Internal errors returned by Service Channel table](#).

Example

```

VAR
  Ret_code:  DWORD;
  ExecMode : INT := MODE_WAIT;
  ExecStatus : DWORD;
  NodeAddress : WORD := 2;
  IDN : DWORD := 51; (* Position feedback value 1 *)
  IDNelement : INT := 7; (* Read current value *)
  ValuesNum : INT;
  Value : DWORD;
  Timeout: DINT := 10000; (* 10 sec *)

  END_VAR

  Ret_code := S3_ReadParameter (ExecMode, ExecStatus, 0, NodeAddress,
                                IDN, IDNelement, ValuesNum, Value,Timeout);

```

See also

[S3_WriteParameter](#)

6.4 S3_WriteDeviceData

The S3_WriteDeviceData function writes information referred to a device connected on a device with SERCOSIII bus.

Syntax

ret_code := S3_WriteDeviceData (BoardNumber, NodeAddress, InfoCode, Value);

Input parameters

<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Node address on the bus [1,128]
<i>InfoCode</i> (WORD)	Information code to write
<i>Value</i> (ANY_ELEMENTARY)	Value to write. This information can be BYTE, WORD or DWORD type.

Use

Function allows to write the following information

<i>InfoCode</i>	Description	Format
1	Device Control Word	WORD
3	Connection Control Word	WORD

The function has to be called within a task with 0 priority.

Return values

Value (HEX)	Description
0	Function executed without errors
000338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00340006	<i>InfoCode</i> parameter out of range

Example

```

VAR
Ret_code: DWORD;
NodeAddress : WORD := 7;
InfoCode: WORD := 1; (* Write control word *)
DeviceData: DWORD;

END_VAR

Ret_code := S3_WriteDeviceData (0, NodeAddress, InfoCode, DeviceData);

```

See also

[S3_ReadDeviceData](#)

6.5 S3_CommStatus

The S3_CommStatus function reads the SERCOSIII communication status.

Syntax

```
ret_code := S3_CommStatus (BoardNumber, CommStatus);
```

Input parameters

BoardNumber (WORD) Board number

Output parameter

CommStatus (STRUCT) This structure contains information referred to SERCOSIII communication status

Use

The function reads information on the communication status of the SERCOSIII field bus. All information are grouped in the S3_CommStatusStruct structure made as follow:

Field	Type	Description
CommPhase	DWORD	SERCOS communication phase
CommErrorPhase	DWORD	SERCOS communication phase error
CommunicationCOS	DWORD	COS (Master)
CommunicationState	DWORD	Current communication status (Master)
CommunicationError	DWORD	Current communication error (Master)
ErrorCount	DWORD	Number of errors from the start (Master)
SlaveState	DWORD	Nodes status (Master)
NumDiags	DWORD	Diagnostic records number available for reading
NumOfConfigSlaves	DWORD	Number of nodes set
NumOfActiveSlaves	DWORD	Active nodes number
NumOfDiagSlaves	DWORD	Number of nodes in diagnostic mode
GeneralFlags	DWORD	Reserved
SlavesDiagMap	ARRAY [0..15] of BYTE	Nodes bit map serious error
CommonErrorReason	DWORD	Internal error reason

Return values

Value (HEX)	Description
0	Function executed without errors
0x00338004	Invalid board, check <i>BoardNumber</i>

Example

```
VAR
  Ret_code: DWORD;
  Status : S3_CommStatusStruct;

  END_VAR

  Ret_code := S3_CommStatus (0, Status);
```

6.6 S3_ReadDiagLog

The S3_ReadDiagLog function reads the diagnostics internal to the communication master.

Syntax

```
ret_code := S3_ReadDiagLog (ExecMode, ExecStatus, BoardNumber, DiagLogEntry, Timeout);
```

Input parameter

<i>ExecMode</i> (INT)	Execution mode [MODE_WAIT, MODE_POLLING]
<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>Timeout</i> (DINT)	Maximum execution time

Output parameters

<i>ExecStatus</i> (DWORD)	Execution status
<i>DiagLogEntry</i> (STRUCT)	This structure contains diagnostic information

Use

The function reads in chronological order the communication master events. Every call returns an S3_DiagLogEntryStruct record type, containing all information referred to an event. The records number currently present can be read with the S3_CommStatus function (field NumDiags of the S3_CommStatusStruct structure).

Field	Type	Description
<i>EntryType</i>	DWORD	Event code
<i>TimestampS</i>	DWORD	Timestamp in seconds
<i>TimestampNs</i>	DWORD	Timestamp in nanoseconds
<i>DataLog</i>	ARRAY [0..9] OF DWORD	Information referred to the event

Below, events codifications with the information associated (not in numerical order).

EntryType = 0x01 New communication phase

<i>DataLog</i>	Associated information
0	New communication phase (0x7f, 0, 1, 2, 3, 4)

EntryType = 0x02 Init command failed

<i>DataLog</i>	Associated information
0	Topology node address in error
1	Node SERCOS address in error
2	IDN parameter causing the error
3	Error cause
4	Service Channel error
5	FSM node status

Possible values causing the error (Datalog[3])

0x0001	Write errors
0x0002	Read error
0x0003	Data comparison failed
0x0004	Service task error
0x0005	Command failed

EntryType = 0x03 Communication error with remote node

<i>DataLog</i>	<i>Associated information</i>
0	Topology address of the node in error
1	SERCOS address of the node in error
2	Error cause
3	Service Channel error
4	FSM node status

Possible values causing the error (Datalog[2])

0x0001	Write errors
0x0002	Read error
0x0003	Data comparison failed
0x0004	Service task error
0x0005	Command failed
0x0006	MHS AHS timeout
0x0007	Timeout, busy node
0x0008	Transition error in CP3
0x0009	Transition error in CP4
0x000A	Node not set
0x000B	Missing node
0x000C	Error when measuring the SYNC delay

EntryType = 0x0C Communication warning with remote node

<i>DataLog</i>	<i>Associated information</i>
0	Topology address of the node in error
1	SERCOS address of the node in error
2	Warning cause
3	Service Channel error
4	FSM node status
5	IDN parameter causing the warning

Possible values causing “warning (Datalog[2])”

0x0001	Missing command acknowledge
--------	-----------------------------

EntryType = 0x08 Bus topology change

<i>DataLog</i>	<i>Associated information</i>
0	Ring interruption

EntryType = 0x09 HotPlug enables

EntryType = 0x0A HotPlug aborted

EntryType = 0x0B HotPlug completed

DataLog	Associated information
0	Address of the node involved in the HotPlug procedure

EntryType = 0x0E **Topology changed without request**

EntryType = 0x10 **Topology change request denied**

EntryType = 0x11 **Topology changed**

DataLog	Associated information
0	Address of the node involved in the topology change

EntryType = 0x0F **Interruption of the topology change request**

DataLog	Associated information
0	Address of the first node involved in the topology change
1	Address of the second node involved in the topology change
2	Ldl error

EntryType = 0x12 **Internal errors**

DataLog	Associated information
0	ID function
1	Error code

EntryType = 0x13 **Transmission length error (S-0-01050.x.5)**

DataLog	Associated information
0	Topology address node
1	Node SERCOS address
2	Connection request
3	Master side connection length
4	Node side connection length

EntryType = 0x06 **Error when initializing the channel**

EntryType = 0x07 **DPM watchdog error**

EntryType = 0x0D **Topology change when measuring the ring delay**

EntryType = 0x14 **Bus scanning request**

EntryType = 0x15 **External trigger timeout**

EntryType = 0x16 **Missed pulse of the external trigger**

EntryType = 0x17 **Communication with all nodes interrupted**

Return values

Values (HEX)	Description
0	Function executed without errors
0x00338003	Invalid execution mode, check <i>ExecMode</i>
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338010	Diagnostic cannot be read
0x00338031	Busy Master
0x00338034	Dwell time of the communication request

Example

```
VAR
  Ret_code:  DWORD;
  ExecMode : INT := MODE_WAIT;
  ExecStatus : DWORD;
  DiagLogEntry: S3_DiagLogEntryStruct;
  Timeout: DINT := 10000; (* 10 sec *)

END_VAR

Ret_code := S3_ReadDiagLog (ExecMode, ExecStatus, 0, DiagLogEntry, Timeout);
```

See also

S3_CommStatus

6.7 S3_WriteParameter

The S3_WriteParameter function writes an IDN parameter of a device connected on SERCOSIII bus.

Syntax

```
ret_code := S3_WriteParameter (ExecMode, ExecStatus, BoardNumber, NodeAddress, ModuleIndex, IDN,
IDNelement, ValuesNum, Values, Timeout);
```

Input parameters

<i>ExecMode</i> (INT)	Execution mode [MODE_WAIT, MODE_POLLING]
<i>BoardNumber</i> (WORD)	Board number [0,1]
<i>NodeAddress</i> (WORD)	Node address on the bus [1..128]
<i>ModuleIndex</i> (WORD)	Modulus index
<i>IDN</i> (DWORD)	IDN parameter to write
<i>IDNelement</i> (INT)	IDN element
<i>ValuesNum</i> (INT)	Number of values to write
<i>Values</i> (ANY)	Values to write. This parameter can be BYTE, WORD, DWORD, ARRAY OF BYTE, ARRAY OF WORD, ARRAY OF DWORD, STRING type
<i>Timeout</i> (DINT)	Maximum execution time

Output parameter

<i>ExecStatus</i> (DWORD)	Execution status
---------------------------	------------------

Use

For the device with *NodeAddress* address, the function writes the information shown in *IDNelement* of an IDN parameter (see S3_ReadParameter function). Usually the information to write is the current parameter value, therefore set *IDNelement*=7.



The current parameter value is expressed in units internal to the drive; for the conversion on its internal unit, read and process the attribute value (see S3_ReadParameter function for the coding).

In MODE_WAIT mode, the function waits for the command to finish for the maximum time indicated by the *Timeout* parameter (expressed in milliseconds). It is advisable to indicate a timeout value long enough (for instance some seconds) to avoid the early output of the function while the command on the device is still running.

In MODE_POLLING mode the function has to be recalled until the *ExecStatus* parameter is equal to COMMAND_RUNNING. In this mode the timeout is managed by the user .

ValuesNum output parameter can have following values:

- ▷ If *Values* parameter is scaled type (that is BYTE, WORD, DWORD) value is 1
- ▷ the *Values* parameter is array type, the number of elements read
- ▷ the *Values* parameter is array type, the number of elements to write

Return values

Value (HEX)	Description
0	Function executed without errors
0x00338003	Invalid execution mode, check <i>ExecMode</i>
0x00338004	Invalid board, check <i>BoardNumber</i>
0x00338030	Node not found, check <i>NodeAddress</i>
0x00338031	Node busy
0x00338032	Node communication errors not recoverable
0x00338034	Communication request dwell time expired
0x00338042	Invalid <i>IDNelement</i>
0x00338043	Data cut (data dimension is too big)

See also the Internal errors returned by **Service Channel table**.

Example

VAR

```
Ret_code: DWORD;
ExecMode : INT := MODE_WAIT;
ExecStatus : DWORD;
NodeAddress : WORD := 2;
IDN : DWORD := 104; (* Position loop Kv value *)
IDNelement : INT:= 7; (* Write current value *)
Value : WORD := 3; (* Internal unit *)
Timeout: DINT := 10000; (* 10 sec *)
```

END_VAR

```
Ret_code := S3_WriteParameter (ExecMode, ExecStatus, 0, NodeAddress,
                               IDN, IDNelement, 1, Value, Timeout);
```

See also

S3_ReadParameter

6.8 Service Channel internal errors table

0x00330001	Service Channel closed
0x00330009	Invalid access when closing the Service Channel
0x00331001	Invalid IDN
0x00331009	Invalid access to element 1
0x00332001	IDN name missing
0x00332002	Short IDN name transmission
0x00332003	Long IDN name transmission
0x00332004	IDN Name cannot be changed (read only)
0x00332005	IDN Name cannot be written at the moment
0x00333002	IDN attribute transmission too short
0x00333003	IDN attribute transmission too long
0x00333004	Attribute cannot be changed (read only)
0x00333005	Attribute cannot be written in this moment
0x00334001	Unit of measure missing
0x00334002	Unit of measure transmission too short
0x00334003	Unit of measure transmission too long
0x00334004	Unit of measure cannot be changed (read only)
0x00334005	Unit of measure cannot be written at the moment
0x00335001	Minimum value missing
0x00335002	Minimum value transmission too short
0x00335003	Minimum value transmission too long
0x00335004	Minimum value cannot be changed (read only)
0x00335005	Minimum value cannot be written at the moment
0x00336001	Minimum value missing
0x00336002	Maximum value transmission too short
0x00336003	Maximum value transmission too long
0x00336004	Maximum value cannot be changed (read only)
0x00336005	Maximum value cannot be written at the moment
0x00337002	Value transmission too short
0x00337003	Value transmission too long
0x00337004	Read only value
0x00337005	Value cannot be written at this communication stage
0x00337006	Value is lower than the minimum allowed
0x00337007	Value exceeds the maximum allowed
0x00337008	Value of the number or of the bit combination is invalid
0x00337009	Value is write-protected by a password
0x0033700A	Value is write-protected
0x0033700B	Invalid indirect addressing (IDN list)
0x0033700C	Value is write protected because of other settings
0x0033700D	Value express in gloating is invalid
0x0033700E	Value is write protected at parameter level
0x0033700F	Value is write protected at operative level
0x00337010	Command already active
0x00337011	Command cannot be stopped
0x00337012	Command cannot be executed at the moment

0x00337013	Invalid parameters: command cannot be executed
------------	--

7. OPEN_AXIS LIBRARY: Axes motion functions

AXD_ReadParameter	Reads a parameter of a digital drive
AXD_RTFeedbackStart	Runs the configuration for monitoring variables of a digital drive.
AXD_WriteParameter	Writes a parameter of a digital drive
AXD_RTFeedbackRead	Reads the real-time variables of a digital drive.
AXD_RTFeedbackStop	Stops the reading of digital drive variables.
AXD_RTTargetStart	Configures the writing process of the digital drive variables
AXD_RTTargetWrite	Writes the real-time variables of a digital drive
AXD_RTTargetStop	Stops the writing process of a digital drive
AX_ReadParameter	Reads the parameters corresponding to an axis
AX_WriteParameter	Changes the parameters corresponding to an axis
AX_ChangeServoMode	Changes servo loop mode
AX_CheckHardwareOvertravel	Manages the hardware limit switches of an axis
AX_SetHardwareOvertravel	Sets the status of the hardware over travel limit switches
AX_Disable	Disables the axis servo loop modes
AX_Enable	Enables the axis servo loop modes
AX_Reset	Is used to abort a command on the axis
AX_ReadStatus	Reads axis status
AX_SetStatus	Sets some status flags of the axis
AX_ReadPendingCommands	Determines the status of axis inner commands
AX_Filter	Configures the filters of an axis
AX_EnableProbe	Enables the storage of the positions of the specified axes when the probing signal is generated
AX_ReadProbePosition	Reads probe values of the axes involve in the probing cycle
AX_ReadProbeStatus	Reads probe status of the axes specified
AX_SetTestMode	Sets the input or the out pin test mode for the axes specified
AX_ResetPosition	Sets the axis position to zero
AX_SelectGearFoller	Allows to select the range for spindle axes or to set the axis servo error type
AX_SpindleSwitch	Allows to change the spindle use mode activating axis characterization
AX_CalibDisable	Disables all compensations generated by an axis.
AX_CalibEnable	Enables an axis in receiving all available compensations
AX_CalibLoadFile	Upload the axis compensation file
AX_CalibDisableGet	Disables for an axis the compensations made by other axes
AX_CalibEnableGet	Enables for an axis the compensations made by other axes
AX_CalibStatus	Returns the Word status active compensations on an axis
AX_ServoToPLC	Allows the servo to communicate with the PLC
AX_PLCToServo	Allows the PLC to reply the servo requests
RS_CreateResource	Allows to set a new d/a, a/d or transducer resource in the servo loop task

RS_DestroyResource	Deletes a D/A, AD resource or a transducer
RS_SensorExecCmd	Executes commands on the sensor
RS_SensorReadPar	Reads sensor parameters
RS_SensorWritePar	Writes sensor parameters
RS_SensorReadTable	Reads the sensor calibration table
RS_SensorWriteTable	Writes the sensor calibration table
RS_TransducerRead	Reads the count of a local or remote encoder transducer on an os-wire bridge
RS_AnalogRead	Reads a local a/d convertor on the axes board
RS_AnalogWrite	Writes a local d/a convertor on the axes board or on the os-wire bridge device

7.1 AXD_ReadParameter

Function AXD_ReadParameter reads a parameter of a digital drive.

Syntax:

```
ret_code := AXD_ReadParameter (AxisId, Param, ParamType, Value) ;
```

Input parameters:

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Param</i> (WORD)	Parameter number
<i>ParamType</i> (INT)	Parameter type

Output parameters:

<i>Value</i> (ANY_ELEMENTARY)	Read value
-------------------------------	------------

Possible overloads:

AXD_ReadParameter (INT,WORD,INT,LREAL)
 AXD_ReadParameter (INT,WORD,INT,STRING)

Execution mode:

Wait

Use:

This function reads the parameters of an OS3 drive associated with an axis. Note for OS3 drives:

- ▷ For codes and formats of the drive parameters, see “OS3 Drive - Installation Manual”.
- ▷ Parameter *ParamType* is not managed at present, force it to 0.

Notes for drives with SERCOSIII communication interface

- ▷ For the parameters format and codification, refer to the drive manual
- ▷ The *ParamType* parameter is not used, set value =0

Notes for drives with SoE communication interface

- ▷ For the parameters format and codification, refer to the drive manual
- ▷ The *ParamType* parameter is not used, set value =0

Notes for drives with Mechatrolink communication interface

- ▷ For the parameters format and codification, refer to the drive manual

In *ParamType* the format parameter must be indicated. Set properly one of the following combinations :

Value (DEC)	Value (HEX)	Description
33	0x21	Integer format not signed expressed on 16 bits
34	0x22	Integer format signed expressed on 16 bits
49	0x31	Integer format not signed expressed on 32 bits
50	0x32	Integer format signed expressed on 32 bits

Return values

The following table lists all possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200001	Insufficient memory for processing
0x00200003	Too request to the axes board
0x00200004	AxisId wrong.
0x00200006	Axis not set
0x00200007	Axes board not set
0x0020000F	Call on analog or virtual axis
0x00200012	Buffers available for the communication finished
0x00200062	Axis in motion (in case of Mechatrolink interface)
0x00200070	Communication timeout with digital axis
0x00200071	Digital axis communication channel busy
0x00200072	Digital axis communication channel disabled for serious error
0x00200073	Digital axis communication phase not consistent
0x00200074	At the moment, the axis is not connected to a digital drive
0x0020008C	Wrong ParamType (where needed).
0x0018xxxx	Errors returned by OS-Wire ambient
0x00220002	Invalid communication buffer (in case of SERCOSIII and SoE interfaces).
0x00260006	Drive alarm (in case of Mechatrolink interface)
0x00260007	Timeout expired while waiting for the feedback (in case of Mechatrolink interface)
0x00260008	Warning parameter (in case of Mechatrolink interface)
0x0033xxxx	Errors returned by SERCOS ambient

Example for OS3 drive:

```
Ret : dword;
val : lreal ;
Ret := AXD_ReadParameter (1, 5003, 0, val);
```

Reads the KV (parameter 5003) of the OS3 drive connected to the axis with ID=1.

See also:

[AXD_WriteParameter](#)

7.2 AXD_RTFeedbackStart

The AXD_RTFeedbackStart function runs the configuration for monitoring variables of a digital drive.

Syntax

```
ret_code := AXD_RTFeedbackStart (AxisId, NumVar, VarId, VarType) ;
```

Input parameters

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>NumVar</i> (INT)	Numbers of variables to monitor [1..4]
<i>VarId</i> (ARRAY OF DWORD)	ID of the variables to be monitored
<i>VarType</i> (ARRAY OF DWORD)	Variables type to be monitored

Execution mode

Wait

Use

This function prepares the system for the variable real-time reading of the drive connected to the axis. Following this function call, variables value can be read using the AXD_RTFeedbackRead function.

Type and number of monitorable variables (up to four) depend from the communication protocol used by the drive. Refer to the following table:

Communication protocol	Number of configurable variables	Description
OS-Wire	4	OS3 drive variables, for the complete list see the Installation Manual and the calibrations
Mechatrolink I e II	1	“Monitor Data” types of the Mechatrolink I and II drives (refer to the Installation Manual)
Mechatrolink III	2	“Monitor Data” types of the Mechatrolink III drives (refer to the Installation Manual)
EtherCAT SoE	4	IDN allocated in cyclical data (to display in the EtherCAT configurator). Only numerical IDN with 2 and 4 bytes length are accepted.



When homing on marker or during the probing cycle, with Mechatrolink I and II, the variable monitored gets a different meaning.

When homing on marker or during the probing cycle, with Mechatrolink III, the first variable monitored gets a different meaning.

Even if currently the *VarType* array is not in use, it has to be set in function input.

Return values

The following table lists all possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200004	AxisId wrong.
0x00200006	Axis not configured
0x0020000F	Call on non-digital axis
0x00200012	Memory resources not available
0x00200020	Monitoring already active
0x00200021	Inactive monitoring
0x00200022	Unexpected function
0x00200023	Not defined variable
0x00200024	Wrong number of variables
0x00200070	Time-out dwell communication channel
0x0021xxxx	OS-Wire ambient errors
0x00220008	IDN type unaccepted (in case of SoE interface)

Example

```
Ret : dword;
VarId : ARRAY[0..3] of dword;
VarType : ARRAY[0..3] of dword; (* Not used *)

VarId[0] := 39; (* measured current *)
VarId[1] := 8; (* NC position target *)
Ret := AXD_RTFeedbackStart (1, 2, VarId, VarType);
```

Prepares the reading of two variables of the OS3 drive associated to the axis with ID=1.

See also

[AXD_RTFeedbackRead](#), [AXD_RTFeedbackStop](#)

7.3 AXD_WriteParameter

Function AXD_WriteParameter writes a digital drive parameters.

Syntax:

```
ret_code := AXD_WriteParameter (AxisId, Param, ParamType, Value) ;
```

Input parameters:

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Param</i> (DWORD)	Parameter number
<i>ParamType</i> (INT)	Parameter type
<i>Value</i> (ANY_ELEMENTARY)	Value to write
<i>Mask</i> (DWORD)	Mask

Possible overloads:

AXD_WriteParameter (INT,WORD,INT,LREAL)
 AXD_WriteParameter (INT,WORD,INT,STRING)

Execution mode:

Wait

Use:

This function makes it possible to write a numerical parameter on a digital drive associated with an axis.

Notes for OS3 drives.

- ▷ For drive parameters codes and format, refer to “OS3 Drive - Installation Manual”.
- ▷ *ParamType* and *Mask* parameters are not used, set value =0

Notes for drives with SERCOSIII communication interface

- ▷ For the parameters format and codification, refer to the drive manual
- ▷ The *ParamType* parameter is not used, set value =0

Notes for drives with SoE communication interface

- ▷ For the parameters format and codification, refer to the drive manual
- ▷ The *ParamType* parameter is not used, set value =0

Notes for drives with Mechatrolink communication interface

- ▷ For the parameters format and codification, refer to the drive manual
- ▷ In *ParamType* the format parameter must be indicated. Set properly one of the following combinations:

Value (DEC)	Value (HEX)	Description
33	0x21	Integer format not signed expressed on 16 bits
34	0x22	Integer format signed expressed on 16 bits
49	0x31	Integer format not signed expressed on 32 bits
50	0x32	Integer format signed expressed on 32 bits

Setting *ParamType* bit 7 at 1 (mask 0x80), the parameter writing will be in the EEPROM drive.

- ▷ In *Mask* it has to be indicated which are the bits parameter that have to be changed. Bits set at 1 will be changed with the new value, bits set at 0 do not change the corresponding value. With *Mask* = 0xFFFFFFFF all parameter bits change.



Changing some parameters can be dangerous, leading to the axis instability.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors.
0x00200001	Insufficient memory to elaborate
0x00200003	Too many requests to axes board
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x00200007	Axes board not configured.
0x0020000F	Call to virtual axis.
0x00200070	Communication with digital axis timeout.
0x00200071	Communication with digital axis channel busy.
0x00200072	Communication with digital axis channel deactivated due to severe error.
0x00200073	Stage of communication with digital axis non-conforming.
0x00200074	Axis not connected to digital drive at present.
0x00200084	Value out of valid range
0x0020008C	<i>ParamType</i> wrong (if needed).
0x0018xxxx	Errors returned from OS-Wire environment.
0x00260007	Timeout expired waiting for the feedback (in case of Mechatrolink interface).
0x00260008	Warning parameter (in case of Mechatrolink interface).
0x0033xxxx	Errors returned from SERCOS ambient

Example for OS3 drive:

```
Ret : dword;
Ret := AXD_WriteParameter (1, 1, 0, 5.0);
```

Sets the ‘Max. following error’ (parameter 1) of the OS3 drive connected to the axis with ID=1 to 5.

See also:

[AXD_ReadParameter](#)

7.4 AXD_Command

The AXD_Command function sends a digital drive command.

Syntax

ret_code := AXD_Command (*AxisId*, *Command*, *Timeout*) ;

Input parameters

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Command</i> (INT)	Command code
<i>Timeout</i> (INT)	Maximum dwell time (milliseconds)

Execution mode

Wait

Use

With this function a command can be sent to a digital drive associated to an axis.

The OS3 drives do not support this function.

Drives with SERCOSIII communication interface cannot support this function, but the S3_Command function could be the proper alternative.

Drives with SoE communication interface cannot support this function, but the ECAT_Command function could be the proper alternative.

Following commands can be sent for drives with Mechatrolink communication interface:

Command (DEC)	Mnemonic
0000	Mech_NOP
0004	Mech_CONFIG
0006	Mech_ALMCLR
0013	Mech_SYNCSET
0015	Mech_DISCONNECT
0033	Mech_BRKON
0034	Mech_BRKOFF
0035	Mech_SESON
0036	Mech_SESOFF
0037	Mech_HOLD
0038	Mech_MLOCKON
0039	Mech_MLOCKOFF



Sending some commands could be dangerous leading to the axis instability.

Return values

The following table lists all possible value that *ret_code* variable can have.

ret_code(HEX)	Description
0x00000000	Function executed without errors
0x00200001	Insufficient memory for processing
0x00200003	Too many requests for axes board
0x00200004	AxisId wrong.
0x00200006	Axis not configured
0x00200007	Axis board not configured
0x0020000F	Call on virtual or analogic axis
0x00200062	Axis in motion (in case of Machatrolink interface)
0x00200070	Timeout of communication with digital axis.
0x00200071	Communication channel with digital axis busy.
0x00200072	Digital axis communication channel disabled for serious error
0x00200073	Communication phase with digital axis not complying
0x00200074	Axis is not connected with a digital drive
0x00200077	Timeout value should be higher than zero
0x00200080	Unexpected Command value
0x00260006	Drive alarm (in case of Machatrolink interface).
0x00260007	Timeout expired waiting for the feedback (in case of Mechatrolink interface).
0x00260008	Warning parameter (in case of Machatrolink interface).

Example for a drive with Mechatrolink interface

```
Ret : dword;
Ret := AXD_Command (1, 6, 1000);
```

Sends the ALM_CLR command to the drive connected to the axis with ID=1all'azionamento and waits for no more than 1 second.

See also

[S3_Command](#), [ECAT_Command](#)

7.5 AXD_RTFeedbackRead

The AXD_RTFeedbackRead function reads the real-time variables of a digital drive.

Syntax

```
ret_code := AXD_RTFeedbackRead (AxisId, VarValue) ;
```

Input parameter

AxisId (INT) Axis identifier [1..64]

Output parameter

VarValue (ARRAY OF LREAL) Values of the monitored variables

Execution mode

Immediate

Use

This function reads in real-time the variables of the digital drive connected to the axis. These variables have to be previously set at reading by the AXD_RTFeedbackStart function.

In case of drives with OS-Wire field bus and EtherCAT SoE interface, variables can be converted; in case of Mechatrolink interface, variables are not converted remaining in the drives unit of measure.

Return values

The following table lists all possible value that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200004	AxisId wrong.
0x00200006	Axis not set
0x0020000F	Call on not digital axis
0x00200012	Memory resources not available
0x00200021	Monitoring disabled
0x00200022	Unexpected function

Example

```
Ret : dword;
VarValue : ARRAY[0..3] of lreal;

Ret := AXD_RTFeedbackRead (1, VarValue);
```

Reads the variables of the drive associated to the axis with ID=1.

See also

[AXD_RTFeedbackStart](#), [AXD_RTFeedbackStop](#)

7.6 AXD_RTFeedbackStop

The AXD_RTFeedbackStop function stops the reading of digital drive variables.

Syntax

```
ret_code := AXD_RTFeedbackStop (AxisId) ;
```

Input parameters

AxisId (INT) Axis id [1..64]

Execution mode

Wait

Use

This function stops the update of the real-time variables monitored by the digital drive connected to the axis. Function has to be used for changing the variables to be monitored before calling for the second time the AXD_RTFeedbackStart function with new variables.

Return values

The following table lists all possible values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200004	AxisId wrong.
0x00200006	Axis not set
0x0020000F	Call on not digital axis
0x00200012	Memory resources not available
0x00200022	Unexpected function

Example

```
Ret : dword;
```

```
Ret := AXD_RTFeedbackStop (1) ;
```

Stops the variables monitoring of the drive associated to axis with ID=1.

See also

[AXD_RTFeedbackStart](#), [AXD_RTFeedbackRead](#)

7.7 AXD_RTTargetStart

The AXD_RTTargetStart function configures the writing process of the digital drive variables.

Syntax

```
ret_code := AXD_RTTargetStart (AxisId, NumVar, VarId, VarType) ;
```

Input parameters

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>NumVar</i> (INT)	Number of variables to write [1..8]
<i>VarId</i> (ARRAY OF DWORD)	ID of the variables to be written
<i>VarType</i> (ARRAY OF DWORD)	Type of the variables to be written

Execution mode

Wait

Use

This function configures the real-time writing of the variables of the digital drive connected to the axis. Following this function call, the variables' values can be written using the AXD_RTTargetWrite function.

The type and number (up to eight) of the variables that can be written depends on the communication protocol used by the drive; refer to the following table:

Communication protocol	Number of configurable variables	Description
EtherCAT CoE	8	objects allocated in the cyclic data (to be set in the EtherCAT configurator). Only numerical objects with 1, 2 and 4 bytes length are accepted. The index object has to be stored in the high word of the variable ID; the objects sub-index has to be stored in the bottom word of the variable ID.
EtherCAT SoE	8	IDNs allocated in the cyclic data (to be set in the EtherCAT configurator). Only numerical IDN with 2 and 4 bytes length are accepted.

For the EtherCAT CoE, the VarType array allows to indicate the parameter format. Properly set one of the following combinations:

Value (DEC)	Value (HEX)	Description
17	0x11	Not signed integer format expressed on 8 bits
18	0x12	Signed integer format expressed on 8 bits
33	0x21	Not signed integer format expressed on 16 bits
34	0x22	Signed integer format expressed on 16 bits
49	0x31	Not signed integer format expressed on 32 bits
50	0x32	Signed integer format expressed on 32 bits

Return values

The following table lists all values the *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	Wrong AxisId.
0x00200006	Axis not set
0x0020000F	Call on a non-digital axis.
0x00200012	Unavailable memory resources.
0x00200020	Write already enabled
0x00200021	Write non-enabled
0x00200022	Unexpected function
0x00200023	Undefined variable
0x00200024	Wrong number of variable
0x00200070	Wait time-out communication channel

Example

```

Ret : dword;
VarId : ARRAY[0..7] of dword;
VarType : ARRAY[0..7] of dword;
NumVar : INT := 1;

VarId[0] := 16#20A40000; (* index 20A4, subindex 0 *)
VarType[0] := 16#21; (* 2 byte, unsigned *)
Ret := AXD_RTTargetStart(1, NumVar, VarId, VarType);

```

Prepares for writing a variable of the CoE drive associated with the axis with ID=1.

The variables has a 20A4 index and a 0 subindex, is 2 bytes long and does not have a sign.

See also

[AXD_RTTargetWrite](#), [AXD_RTTargetStop](#)

7.8 AXD_RTTargetWrite

The AXD_RTTargetWrite function writes the real-time variables of a digital drive.

Syntax

```
ret_code := AXD_RTTargetWrite (AxisId, VarValue) ;
```

Input parameter

AxisId (INT) Axis identifier [1..64]

Output parameters

VarValue (ARRAY OF LREAL) Value of the variables to be written

Execution mode

Immediate

Use

This function writes in real-time the variables of the digital drive connected to the axis. The variables have to be previously set to be written using the AXD_RTTargetStart function.

See instructions below for the different protocols:

Communication protocol	Instructions
EtherCAT SoE	The values have to be written in the measure unit shown (for instance: mm, RPM, etc.).

Return values

The following table lists all values that the *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	Wrong <i>AxisId</i> .
0x00200006	Axis not set
0x0020000F	Call on a non-digital axis.
0x00200021	Write non-enabled
0x00200022	Unexpected function
0x00200025	Variable value is out of range

Example

```
Ret : dword;
VarValue : ARRAY[0..7] of lreal;
```

```
Ret := AXD_RTTargetWrite (1, VarValue);
```

Writes the variables of the drive associated with an axis having ID=1.

See also

[AXD_RTTargetStart](#), [AXD_RTTargetStop](#)

7.9 AXD_RTTargetStop

The AXD_RTTargetStop function stops the writing process of a digital drive.

Syntax

```
ret_code := AXD_RTTargetStop (AxisId) ;
```

Input parameters

<i>AxisId</i> (INT)	Axis identifier [1..64]
---------------------	-------------------------

Execution mode

Wait

Use

This function stops the real-time variables writing process of the digital device connected to the axis. This function can be used for changing the variables to be written before the second call to the AXD_RTTargetStart function with new variables.

Return values

The following table lists all values that the *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	Wrong <i>AxisId</i> .
0x00200006	Axis not set
0x0020000F	Call on a non-digital axis
0x00200012	Memory resources unavailable
0x00200022	Unexpected function

Example

```
Ret : dword;
```

```
Ret := AXD_RTTargetStop (1);
```

Stops the writing of the drive variables associated with the axis having ID=1. .

See also

[AXD_RTTargetStart](#), [AXD_RTTargetRead](#)

7.10 AX_ReadParameter

Function AX_ReadParameter reads the parameters corresponding to an axis.

Syntax:

```
ret_code := AX_ReadParameter (AxisId, Param, Value) ;
```

Input parameters:

<i>AxisId</i> (INT)	axis identifier [1..64]
<i>Param</i> (INT)	parameter number [1..114]

Output parameter:

<i>Value</i> (LREAL)	parameter value
----------------------	-----------------

Execution mode:

Immediately or Wait depending on parameter to be read.

Use:

This function reads the parameters corresponding to an axis. The table describing the parameters is given below. The function is wait type if the parameter has to be written from the servo loop real-time process.

Param	Mnemonic name	Unit of measure	Reading only	Writing mode	Reading mode
1	fn_AXTYPE	(*)	Yes	-	Immediate
2	fn_AXCLOCK	msec	Yes	-	Immediate
3	fn_CLOCKRAT	(*)	Yes	-	Immediate
4	fn_DIGITAL_AXIS	(*)	Yes	-	Immediate
5	fn_LINEAR_OPTICAL_ENCODER	(*)	No	Wait	Immediate
6	fn_CHANNEL_A_POLARITY_INV	(*)	No	Wait	Immediate
7	fn_CHANNEL_B_POLARITY_INV	(*)	No	Wait	Immediate
8	fn_CHANNEL_Z_POLARITY_INV	(*)	No	Wait	Immediate
9	fn_DIRECTION_COUNT	(*)	No	Wait	Immediate
10	fn_MARKER_DETECTION	(*)	No	Wait	Immediate
11	fn_RAPID_TRAVERSE_FEED	M.U. / min	No	Immediate	Immediate
12	fn_RAPID_ACCELERATION	M.U. / sec ²	No	Immediate	Immediate
13	fn_MANUAL_FEED	M.U. / min	No	Immediate	Immediate
14	fn_MANUAL_ACCELERATION	M.U. / sec ²	No	Immediate	Immediate
15	fn_RAPID_JERK	M.U. / sec ³	No	Immediate	Immediate
16	fn_WORKING_JERK	M.U. / sec ³	No	Immediate	Immediate
17	fn_ELECTRICAL_PITCH	impulses	No	Wait	Immediate
18	fn_MECHANICAL_PITCH	M.U.	No	Wait	Immediate
19	fn_ROLLOVER_PITCH	M.U.	No	Immediate	Immediate
20	fn_RAPID_TRAVERSE_VOLTAGE	Volt	No	Wait	Immediate
21	fn_MAXIMUM_FEED	M.U. / min	No	Wait	Immediate
22	fn_HOME_POSITION_FEED	M.U. / min	No	Wait	Immediate
23	fn_NULL_OFFSET_VALUE	M.U.	No	Wait	Immediate
24	fn_HOME_POSITION_VALUE	M.U.	No	Wait	Immediate

25	fn_HOMING_DIRECTION	(*)	No	Wait	Immediate
26	fn_PERCENT_OF_VFF	%	No	Wait	Immediate
27	fn_UPPER_SW_OVERTRAVEL	M.U.	No	Immediate	Immediate
28	fn_LOWER_SW_OVERTRAVEL	M.U.	No	Immediate	Immediate
29	fn_SERVO_LOOP_GAIN	IPM / mil	No	Wait	Immediate
30	fn_STAND_STILL_LOOP_GAIN	IPM / mil	No	Wait	Immediate
31	fn_POS_ERROR_STAND_STILL	M.U.	No	Wait	Immediate
32	fn_POS_ERROR_WITH_VFF	M.U.	No	Wait	Immediate
33	fn_POS_ERROR_WITHOUT_VFF	M.U.	No	Wait	Immediate
34	fn_IN_POSITION_BAND	M.U.	No	Wait	Immediate
35	fn_IN_POSITION_WAIT	sec	No	Wait	Immediate
36	fn_IN_POSITION_WINDOW	sec	No	Wait	Immediate
37	fn_AXIS_BACKLASH	M.U.	No	Wait	Immediate
38	fn_DEAD_ZONE	M.U.	No	Wait	Immediate
39	fn_DRIVER_ADDRESS	Address	Yes	-	Immediate
40	fn_HOMING_TYPE	(see ODM)	No	Wait	Immediate
41	fn_ADDITIONAL_SERVICE	(see ODM)	No	Wait	Immediate
42	fn_MOTOR_TRANSDUCER	(*)	Yes	-	Immediate
43	fn_PROBING_CONFIGURATION	(see ODM)	No	Wait	
44	fn_SPINDLE_ZERO_OFFSET	deg	No	Immediate	Immediate
45	fn_SPINDLE_WITH_RAMPS	(*)	Yes	-	Immediate
46	fn_SPINDLE_CSS_MASTER	Axis Id	No	Immediate	Immediate
47	fn_SPINDLE_REV_TIME_GEAR1	sec	No	Immediate	Immediate
48	fn_SPINDLE_REV_TIME_GEAR2	sec	No	Immediate	Immediate
49	fn_SPINDLE_REV_TIME_GEAR3	sec	No	Immediate	Immediate
50	fn_SPINDLE_REV_TIME_GEAR4	sec	No	Immediate	Immediate
51	fn_SPIN_POSERR_STANDSTILL	deg	No	Wait	Immediate
52	fn_SPINDLE_SPEED_GEAR1	RPM	No	Wait	Immediate
53	fn_SPINDLE_SPEED_GEAR2	RPM	No	Wait	Immediate
54	fn_SPINDLE_SPEED_GEAR3	RPM	No	Wait	Immediate
55	fn_SPINDLE_SPEED_GEAR4	RPM	No	Wait	Immediate
56	fn_SPINDLE_VOLTAGE_GEAR1	Volt	No	Wait	Immediate
57	fn_SPINDLE_VOLTAGE_GEAR2	Volt	No	Wait	Immediate
58	fn_SPINDLE_VOLTAGE_GEAR3	Volt	No	Wait	Immediate
59	fn_SPINDLE_VOLTAGE_GEAR4	Volt	No	Wait	Immediate
60	fn_SPINDLE_GAIN_GEAR1	IPM / mil	No	Wait	Immediate
61	fn_SPINDLE_GAIN_GEAR2	IPM / mil	No	Wait	Immediate
62	fn_SPINDLE_GAIN_GEAR3	IPM / mil	No	Wait	Immediate
63	fn_SPINDLE_GAIN_GEAR4	IPM / mil	No	Wait	Immediate
64	fn_OFFSET_BETWEEN_MARKERS	impulses	No	Wait	Wait
65	fn_SKEW_ERROR_MIN	M.U.	No	Wait	Wait
66	fn_SKEW_ERROR_MAX	M.U.	No	Wait	Wait
67	fn_SKEW_GAIN	(*)	No	Wait	Wait
68	fn_DRIVE_STATUSW	(*)	Yes	-	Immediate
69	fn_OS3_DRIVE_WARNINGS	(*)	Yes	-	Wait
70	fn_AXIS_IOSTAT	(*)	No	Wait	Wait
71	fn_ACT_MICRO_MARKER_DIST	M.U.	Yes	-	Immediate
72	fn_MAX_MICRO_MARKER_DIST	M.U.	No	Wait	Immediate

73	fn_PROCESS_NUMBER	(*)	Yes	-	Immediate
74	fn_AXIS_SHARING	(*)	Yes	-	Immediate
75	fn_TRANSDUCER_CONFIG	(*)	No	Wait	Immediate
76	fn_ACTIVE_GEAR_NUMBER	#GEAR (0-4)	Yes	-	Immediate
77	fn_ZEROSHIFT1	M.U.	No	No Wait	Immediate
78	fn_ZEROSHIFT2	M.U.	No	No Wait	Immediate
79	fn_AXIS_BACKLASH_TIME	msec	No	Wait	Immediate
80	fn_TRANSDUCER_ID	(*)	Yes	-	Immediate
81	fn_CONVERTER_ID	(*)	Yes	-	Immediate
82	fn_SETRESRIF	(*)	No	Wait	Immediate
83	fn_RAPID_TIME_MIN_RAMP	msec	No	Immediate	Immediate
84	fn_WORKING_TIME_MIN_RAMP	msec	No	Immediate	Immediate
85	fn_RAPID_DECELERATION	M.U. / sec ²	No	Immediate	Immediate
86	fn_MANUAL_DECELERATION	M.U. / sec ²	No	Immediate	Immediate
87	fn_RAMP_MODALITY	(*)	No	Immediate	Immediate
88	fn_GET_MASTER_SLAVE	(*)	Yes	-	Immediate
89	fn_AXIS_INTERFACE_TYPE	(*)	Yes	-	Immediate
90	fn_BW_ENABLE	sec	No	Wait	Immediate
91	fn_BW_TIME	(*)	No	Wait	Immediate
92	fn_EDGE_LEAP_FEED	M.U. / min	No	Immediate	Immediate
93	fn_GET_LAST_ERROR	(*)	Yes	-	Immediate
94	fn_SPINDLE_SS_GAIN_GEAR1	IPM / mil	No	Wait	Immediate
95	fn_SPINDLE_SS_GAIN_GEAR2	IPM / mil	No	Wait	Immediate
96	fn_SPINDLE_SS_GAIN_GEAR3	IPM / mil	No	Wait	Immediate
97	fn_SPINDLE_SS_GAIN_GEAR4	IPM / mil	No	Wait	Immediate
98	fn_SPIN_ORIENT_SPEED_GEAR1	RPM	No	Immediate	Immediate
99	fn_SPIN_ORIENT_SPEED_GEAR2	RPM	No	Immediate	Immediate
100	fn_SPIN_ORIENT_SPEED_GEAR3	RPM	No	Immediate	Immediate
101	fn_SPIN_ORIENT_SPEED_GEAR4	RPM	No	Immediate	Immediate
102	fn_SPIN_ORIENT_ACCEL_GEAR1	rev / sec ²	No	Immediate	Immediate
103	fn_SPIN_ORIENT_ACCEL_GEAR2	rev / sec ²	No	Immediate	Immediate
104	fn_SPIN_ORIENT_ACCEL_GEAR3	rev / sec ²	No	Immediate	Immediate
105	fn_SPIN_ORIENT_ACCEL_GEAR4	rev / sec ²	No	Immediate	Immediate
106	fn_SPIN_STOP_THRESH_GEAR1	RPM	No	Immediate	Immediate
107	fn_SPIN_STOP_THRESH_GEAR2	RPM	No	Immediate	Immediate
108	fn_SPIN_STOP_THRESH_GEAR3	RPM	No	Immediate	Immediate
109	fn_SPIN_STOP_THRESH_GEAR4	RPM	No	Immediate	Immediate
110	fn_SPIN_IN_POS_BAND	deg	No	Wait	Immediate
111	fn_SPIN_IN_POS_WAIT	sec	No	Wait	Immediate
112	fn_SPIN_IN_POS_WINDOW	sec	No	Wait	Immediate
113	fn_ENABLE_SERVO_MODE	(*)	No	Wait	Immediate
114	fn_SKEW_RECOVERY_VEL	M.U. / min	No	Wait	Immediate
115	fn_DRIVE_CONTROLW	(*)	No	Wait	Wait
116	fn_MIN_AX_VELOCITY	M.U. / min	No	Immediate	Immediate
117	fn_MIN_AX_POSITION	M.U.	No	Immediate	Immediate
118	fn_DEN_MEC_PITCH	M.U.	No	Wait	Immediate
119	fn_ZEROSHIFT1_SETUP	M.U.	No	Immediate	-
120	fn_ZEROSHIFT2_SETUP	M.U.	No	Immediate	-

121	fn_RAPID_DJERK	M.U. / sec ³	No	Immediate	Immediate
122	fn_WORKING_DJERK	M.U. / sec ³	No	Immediate	Immediate
123	fn_RAPID_TIME_MIN_DRAMP	msec	No	Immediate	Immediate
124	fn_WORKING_TIME_MIN_DRAMP	msec	No	Immediate	Immediate
125	fn_REFERENCE_OFFSET	Volt	No	Immediate	Immediate
126	fn_EMERG_DECELERATION	M.U. / sec ²	No	Immediate	Immediate
127	fn_UPPER_SW_OVERT_TTIP	M.U.	No	Immediate	Immediate
128	fn_LOWER_SW_OVERT_TTIP	M.U.	No	Immediate	Immediate
129	fn_DRIVER_SUBADDRESS	Index	Si	-	Immediate
130	Fn_INTERP_POINT	M.U.	No	Immediate	-
131	Fn_LEAP_ACCE	M.U. / sec ²	No	Immediate	Immediate

M.U. stands for “Measuring Unit” and, depending on axis characterisation, it may mean millimetres, inches or degrees. The symbol (*) indicates the presence of special notes given below.

Parameter *fx_AXISTYPE* is a mask with bits indicating the main characteristics of an axis (for a better understanding, it is advisable to convert it to hexadecimal whole format). The meaning of the bits is given in the following table:

Bit	Val Hex	Mnemonic	Meaning
0	0x0001	AX_LINEAR	Linear axis
1	0x0002	AX_PLA	Axis of logic
2	0x0004	AX_ROTARY	Revolving axis
3	0x0008		reserved
4	0x0010	AX_SPINDNOTRASD	Spindle without transducer
5	0x0020	AX_SPINDRTRASD	Spindle with transducer
6	0x0040	AX_DIAMETRAL	Diametrical axis
7	0x0080		reserved
8	0x0100	AX_VIRTUAL	Virtual axis
9	0x0200	AX_OPTICAL	Axis with optical line
10	0x0400		reserved
11	0x0800	AX_SPINDRAMP	Spindle with ramp
12	0x1000	AX_GANTRY_MASTER	Axis Gantry Master
13	0x2000	AX_GANTRY_SLAVE	Axis Gantry Slave
14	0x4000	AX_ROLLOVER	Rollover axis
15	0x8000	AX_DIGITAL	Digital axis

Parameter *fn_CLOCKRAT* is a whole number indicating the ratio between the frequency of the clock of the associated interpolator and the frequency of the axis sampling clock.

The DIGITAL_AXIS parameter may assume the following values: 0 (NO), 1 (YES).

Parameter *fn_LINEAR_OPTICAL_ENCODER* parameter may assume the following values: 0 (NO), 1 (YES). For HOMING purposes, process axis flags are read at the start of each interpolation, for logic axes they are read only during the OPEN phase. For virtual axes, this function is ignored.

Parameters *fn_CHANNEL_A_POLARITY_INV*, *fn_CHANNEL_B_POLARITY_INV*, *fn_CHANNEL_Z_POLARITY_INV* and *fn_DIRECTION_COUNT* may assume the following values: 0 (NO), 1 (YES). It is advisable to disable the axis completely before inverting the polarity of the channels or the count direction of the encoder. This function is ignored for axes with no transducer associated, virtual axes, and digital axes.

Parameter *fn_MARKER_DETECTION* may assume the following values: 0 (LEVEL), 1 (EDGE). It is not advisable to change marker edge type during the HOMING stage. This function is ignored for: axes with no transducer associated, virtual axes, digital axes.

For parameters *fn_RAPID_TRAVERSE_FEED*, *fn_RAPID_ACCELERATION*, *fn_MANUAL_FEED*, *fn_MANUAL_ACCELERATION*, *fn_RAPID_JERK* and *fn_WORKING_JERK*, for purposes of the movement, the values are considered at the start of the next interpolation. No range check is made.

For parameters *fn_ELECTRICAL_PITCH* and *fn_MECHANICAL_PITCH* it is advisable to disable the axis completely before calling this function. All the parameters associated with the electrical and/or mechanical pitch are recalculated. The function is ignored for: axes with no transducer associated, virtual axes. For split axes, a call to the Master affects the Slave too.

Before changing the *fn_ROLLOVER_PITCH* parameter, it is advisable to disable the axis completely before calling the function.

A change to the *fn_RAPID_TRAVERSE_VOLTAGE* and *fn_MAXIMUM_FEED* parameters may affect the Servo Error variable algorithm. This function is ignored for: virtual axes, digital axes. It is advisable to disable the axis completely before calling the function.

Parameter *fn_HOME_POSITION_FEED* is considered at the beginning of the following homing. No range check is made.

Parameters *fn_NULL_OFFSET_VALUE* and *fn_HOME_POSITION_VALUE* are actualised at the end of the next reset. It is not advisable to call this function during the HOMING stage. For diametric axes, the write value is halved. No range check is made.

Parameter *fn_HOMING_DIRECTION* can have following values: 0 (POSITIVE), 1 (NEGATIVE). The value is considered at the beginning of the next HOMING. For virtual axes the function is not considered.

Parameter *fn_PERCENT_OF_VFF* is taken into account at the start of the next HOMING stage. For virtual axes this function is ignored.

Parameters *fn_UPPER_SW_OVERTRAVEL* and *fn_LOWER_SW_OVERTRAVEL* are taken into account at the start of the next interpolation. No range check is made. Writing either parameter to zero disables the control of working limits. Writing any value other than zero on either parameter activates the working limit control function. For diametric axes, the value written is halved. For virtual axes this function is ignored.

A change of the *fn_SERVO_LOOP_GAIN* parameter may affect the Servo Error variable algorithm. For virtual axes this function is ignored.

Parameter *fn_STAND_STILL_LOOP_GAIN* is taken into account at the end of the interpolation. For virtual axes this function is ignored.

Parameter *fn_POS_ERROR_STAND_STILL* is taken into account when the axis stops. For virtual axes this function is ignored.

Parameter *fn_POS_ERROR_WITH_VFF* is actualised immediately if the axis is in movement with VFF.

Parameter *fn_POS_ERROR_WITHOUT_VFF* is actualised immediately if the axis is in movement without VFF.

Parameter *fn_IN_POSITION_BAND* is actualised at the end of the interpolation.

Parameters *fn_IN_POSITION_WAIT* and *fn_IN_POSITION_WINDOW* are actualised at the end of the interpolation. No range check is made.

Parameter *fn_AXIS_BACKLASH* is actualised if the axis has been referred to zero. A change to this parameter affects the "Soft" algorithm for the application of mechanical backlash.

No range check is made when the *fn_DEAD_ZONE* parameter is changed.

Parameter *fn_HOMING_TYPE*. For digital axes, the value is taken into account at the start of the next HOMING stage.

No range check is made when the *fn_ADDITIONAL_SERVICE* parameter is changed.

Parameter *fn_MOTOR_TRANSDUCER* may have the following values : 0 (NO) , 1 (YES)

No range check is made when the *fn_PROBING_CONFIGURATION* parameter is changed. The field only concerns the programming of the probing mode for digital drives with D.S.I. interface.

When changing *fn_PROBING_CONFIGURATION* parameter there is no range control. The field only concerns the probe mode programming of the digital drives with D.S.I. interface.

Parameter *fn_OFFSET_FOR_SPINDLE_ORIENT* is taken into account at the start of the spindle orientation stage. No range check is made.

Parameter *fn_SPINDLE_WITH_RAMPS* may have values 0 (NO), 1 (YES) according to the angle active.

Reversal time is set in parameters *fn_SPINDLE_REVERSAL_TIME_GEAR1/2/3/4*. If the value written is other than zero, the acceleration/deceleration ramp is activated according to the value written; if the value written is zero, the transition from one speed to another is performed with a step. The presence or absence of a ramp, depending on the active range, is denoted by bit 11 "Spindle with ramp" of parameter AXIS TYPE and parameter SPINDLE WITH RAMPS.

For parameters *fn_SPINDLE_SPEED_GEAR1/2/3/4*, *fn_SPINDLE_VOLTAGE_GEAR1/2/3/4* and *fn_SPINDLE_GAIN_GEAR1/2/3/4* the function is ignored for virtual and digital axes. It is advisable to disable the spindle completely before calling this function.

For parameters *fn_OFFSET_BETWEEN_MARKERS*, *fn_SKEW_ERROR_MIN* and *fn_SKEW_ERROR_MAX* the function is ignored for virtual axes. It must be used solely on the Master Split axis.

For parameter *fn_SKEW_GAIN* the function is ignored for virtual axes. It must be used solely on the Master Split axis. No range check is made.

For the *fn_DRIVE_STATUSW* parameter encoding for OS3 drives, word status may assume the following values:

Bit	Val Hex	Mnemonic	Meaning
0..3	0x000X	ST 0..3	Current code status.
4	0x0010	DR_STOP	Fixed axis
5	0x0020	DR_AXREF	Axis referred
6	0x0040	DR_BRAKE	Brake status (0 means brake released or not present)
7	0x0080	DR_HOME	Homing process underway
8	0x0100	DR_ZERO_SW	Zero micro input status
9	0x0200	DR_AUTOTRG	Execution Cmd Point2Point , StepCurr or StepVel
10	0x0400	DR_FFW_ON	Enabled Feed Forward (speed and acceleration)
11			Reserved
12	0x1000	DR_ALARM	Emergency signal presence
13	0x2000	DR_SERVO	Servo Loop are active
14	0x4000	DR_POWER	There is the power: Vbus OK
15	0x8000	DR_MARKER	Found Marker

OS3 drives encoding status:

ST3	ST2	ST1	ST0	Mnemonic	Status
0	0	0	0	DR_START	DRIVE START
0	0	0	1	DR_DIAG	DIAGNOSTIC
0	0	1	0	DR_DRVOFF	DRIVE_OFF
0	0	1	1	DR_DRVON	DRIVE_ON
0	1	0	0	DR_HOLD	HOLD
0	1	0	1	DR_SHTDWN	SHUT_DOWN
0	1	1	0	DR_SHUTTFIX	SHUT_TFIX
0	1	1	1	DR_SHUTIGBT	SHUT_IGBT
1	0	0	0	DR_HALT	HALT

For further details see "OS3 Drive - Installation manual" .

For the *fn_OS3_DRIVE_WARNINGS* parameter encoding, see the following table (for further details see OS3 Drive - Installation Manual):

Value	Name	Meaning
0	DR_W_NONE	No warning active
1	DR_W_HEATSINK	Heat sink overheat Warning
2	DR_W_BUSOVERV	Bus Overvoltage Warning
4	DR_W_BUSUNDERV	Bus Undervoltage Warning
8	DR_W_RAPIDHALT	Rapid Halt Warning

For the *fn_AXIS_IOSTAT* parameter encoding, see the following tables.

Possible values read by an OS3 drive:

Bit	Name	Meaning
0	DR_GPI2	General Purpose Input 2
1	DR_GPI1	General Purpose Input 1
2	DR_MICROINP	Input Micro of zero
3	DR_RHALTINP	Input di Rapid Halt
4	DR_GP03	General Purpose Output 3
5	DR_GP02	General Purpose Output2
6	DR_GP01	General Purpose Output 1
7	DR_DRDYOUT	Output di Drive Ready
8	DR_BRAKEOUT	Output Motor Brake
9..15		Not used

(for further details see “OS3 Drive - Installation Manual”)

Possible values read by a Bridge device:

Bit	Name	Meaning
0	BR_GPI1	General Purpose Input 1
1	BR_GPI2	General Purpose Input 2
2	BR_GPI3	General Purpose Input 3
3	BR_OUTPUT_FAULT	When it is 0 it indicates a short-circuit in the digital outputs
4	BR_WATCH_DOG	When it is 0 it indicates that a watch dog for the device is currently active

N.B. In order to read the inputs, it has to be declared an analog axis with transducer or converter mapped on the bridge and the axis ID has to be used as function input.

Parameter *fn_MAXIMUM_MICRO_MARKER_DIST* is taken in consideration at the beginning of homing procedure.

Parameter *fn_PROCESS_NUMBER* is 0 in the case of a logic axis, it is -1 if the axis has been parked, otherwise it is a value between 1 and 24 reflecting the applicable process.

For coordinated axes, the parameter *fn_AXIS_SHARING* may assume the following values: 0 (not shared), 1 (axis shared between process and logic); for spindle axes: 0 (spindle not associated with any process), 1 (spindle shared by several processes), 2 (spindle exclusive to a process).

Parameter *fn_ACTIVE_GEAR_NUMBER* returns values from 1 to 4 for spindle axes and 0 for non-spindle axes.

Values *fn_ZEROSHIFT1* and *fn_ZEROSHIFT2* are interpreted as absolute values to be forced in the zero shift registers of the servo processor modulus. The offsets are actualised regardless of whether or not an axis has already been referred. WinPLUS is able to move the selected axis by means of this function. The current axis position register and the request position registers will not change. Zero shift offsets may be used to compensate for axis errors (e.g., temperature) or to bring about a physical movement of the origin. An offset can be removed through the AX_ResetPosition function.

The offset value contained in the zero shift register can be initialized through the function *AX_ZEROSHIFT1_SETUP* and *AX_ZEROSHIFT2_SETUP*, passing as parameter the initialization value; this value will be bound to the zero shift subtracting the axis position. Passing to 0.0 value, the zero shift contribution is null as happened with the function *AX_ResetPosition*.

For parameters *fn_TRANSDUCER_ID* and *fn_CONVERTER_ID* see the notes in the preface of library manual.

Parameter *fn_SETRERIF* take values 0 (RESRIF) and 1 (SETRIF). It can only be written; to determine the axis referred condition, use function *AX_ReadStatus*.

Parameter *fn_RAMP_MODALITY* indicates the acceleration ramp modality.

Parameter *fn_GET_MASTER_SLAVE_ID* indicates the ID of the complementary axis in the case of Gantry axes.

Parameter *fn_AXIS_INTERFACE_TYPE* indicates one of the following types of axis interface:

Value	Name	Meaning
0	ANALOG_INTERF	Analogic interface, used by analogic axes and axes on Os-Wire Bridge
1	SERCOS_INTERF	SERCOS interface
2	MECHAT_INTERFACE	Mechatrolink interface
3	OSWIRE_INTERF	OS-Wire interface, used by digital axes with OS3 drive
4	CANOPE_INTERF	CANOpen interface, used by digital axes with VFD drive
5	CANA10_INTERF	For analog axes with A10 CANOpen analog modulus
6	CANP24_INTERF	For analog axes with P24 CANOpen analog modulus
8	ETCSOE_INTERF	For digital axes with drive having a SERCOS Over EtherCAT profile
9	ETCNLG_INTERF	For analog axes with EtherCAT PE E-501 analog modulus

Parameter *fn_BW_ENABLE* may have the following values: 0 (NO), 1 (YES).

Parameter *fn_BW_TIME* is taken into account when the axis is enabled.

Parameter *fn_GET_LAST_ERROR* reads the real-time error code of the axis.

For parameters *fn_SPINDLE_SS_GAIN_GEAR1/2/3/4* the function becomes wait type if the change is made on the active range.

Parameters *fn_SPIN_ORIENT_SPEED_GEAR1/2/3/4* and *fn_SPIN_ORIENT_ACCELERATION_GEAR1/2/3/4* are considered at the start of the spindle orientation stage. No range check is performed.

Parameters *fn_SPIN_STOP_THRES_GEAR1/2/3/4* is taken into consideration at the start of the boring cycle. No range check is performed.

Parameter *fn_ENABLE_SERVO_MODE* indicates the servo mode applied to the axis by the enable command (function *AX_ENABLE* with resource 1). The admissible values are those associated with the Primary Operating Mode and the Secondary Operating Mode of the axis (see function *AX_ReadStatus*, status word 2, masks *AX_MODE1MSK* and *AX_MODE2MSK*).

Parameter *fn_SKEW_RECOVERY_VEL* indicates the speed of the Slave Split axis during the realignment stage. When this parameter is written its value must not exceed manual feed rate, or an error will occur.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors.
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Call to virtual axis.
0x00200031	Call to split slave axis.
0x00200080	Param not valid.
0x0020008d	Axis without converter

Example:

```
Ret : dword;
PARM_VALUE : lreal;
Ret := AX_ReadParameter (1, fn_SERVO_LOOP_GAIN, PARM_VALUE);
```

Reads the SERVO LOOP GAIN value of the axis with ID=1.

See also:

[AX_WriteParameter](#), [AX_ReadStatus](#)

7.11 AX_WriteParameter

Function AX_WriteParameter writes the parameters corresponding to an axis.

Syntax:

```
ret_code := AX_WriteParameter (AxisId, Param, Value) ;
```

Input parameters:

<i>AxisId</i> (INT)	axis identifier [1..64]
<i>Param</i> (INT)	parameter number [1..114]
<i>Value</i> (LREAL)	parameter value

Execution mode:

Immediately, Wait or NoWait depending on parameter to be written.

Use:

This function changes the parameters corresponding to an axis. Some parameters cannot be written (check the field "Only reading"), but it is possible to read through AX_ReadParameter. The function is wait type if the parameter has to be written by the servo loop real-time procedure.

Below the table for parameters description:

Param	Mnemonic name	Unit of measure	Only reading	Writing mode	Reading mode
1	fn_AXTYPE	(*)	Yes	-	Immediate
2	fn_AXCLOCK	msec	Yes	-	Immediate
3	fn_CLOCKRAT	(*)	Yes	-	Immediate
4	fn_DIGITAL_AXIS	(*)	Yes	-	Immediate
5	fn_LINEAR_OPTICAL_ENCODER	(*)	No	Wait	Immediate
6	fn_CHANNEL_A_POLARITY_INV	(*)	No	Wait	Immediate
7	fn_CHANNEL_B_POLARITY_INV	(*)	No	Wait	Immediate
8	fn_CHANNEL_Z_POLARITY_INV	(*)	No	Wait	Immediate
9	fn_DIRECTION_COUNT	(*)	No	Wait	Immediate
10	fn_MARKER_DETECTION	(*)	No	Wait	Immediate
11	fn_RAPID_TRAVERSE_FEED	M.U. / min	No	Immediate	Immediate
12	fn_RAPID_ACCELERATION	M.U. / sec ²	No	Immediate	Immediate
13	fn_MANUAL_FEED	M.U. / min	No	Immediate	Immediate
14	fn_MANUAL_ACCELERATION	M.U. / sec ²	No	Immediate	Immediate
15	fn_RAPID_JERK	M.U. / sec ³	No	Immediate	Immediate
16	fn_WORKING_JERK	M.U. / sec ³	No	Immediate	Immediate
17	fn_ELECTRICAL_PITCH	impulses	No	Wait	Immediate
18	fn_MECHANICAL_PITCH	M.U.	No	Wait	Immediate
19	fn_ROLLOVER_PITCH	M.U.	No	Immediate	Immediate
20	fn_RAPID_TRAVERSE_VOLTAGE	Volt	No	Wait	Immediate
21	fn_MAXIMUM_FEED	M.U. / min	No	Wait	Immediate
22	fn_HOME_POSITION_FEED	M.U. / min	No	Wait	Immediate
23	fn_NULL_OFFSET_VALUE	M.U.	No	Wait	Immediate
24	fn_HOME_POSITION_VALUE	M.U.	No	Wait	Immediate

25	fn_HOMING_DIRECTION	(*)	No	Wait	Immediate
26	fn_PERCENT_OF_VFF	%	No	Wait	Immediate
27	fn_UPPER_SW_OVERTRAVEL	M.U.	No	Immediate	Immediate
28	fn_LOWER_SW_OVERTRAVEL	M.U.	No	Immediate	Immediate
29	fn_SERVO_LOOP_GAIN	IPM / mil	No	Wait	Immediate
30	fn_STAND_STILL_LOOP_GAIN	IPM / mil	No	Wait	Immediate
31	fn_POS_ERROR_STAND_STILL	M.U.	No	Wait	Immediate
32	fn_POS_ERROR_WITH_VFF	M.U.	No	Wait	Immediate
33	fn_POS_ERROR_WITHOUT_VFF	M.U.	No	Wait	Immediate
34	fn_IN_POSITION_BAND	M.U.	No	Wait	Immediate
35	fn_IN_POSITION_WAIT	sec	No	Wait	Immediate
36	fn_IN_POSITION_WINDOW	sec	No	Wait	Immediate
37	fn_AXIS_BACKLASH	M.U.	No	Wait	Immediate
38	fn_DEAD_ZONE	M.U.	No	Wait	Immediate
39	fn_DRIVER_ADDRESS	Address	Yes	-	Immediate
40	fn_HOMING_TYPE	(see ODM)	No	Wait	Immediate
41	fn_ADDITIONAL_SERVICE	(see ODM)	No	Wait	Immediate
42	fn_MOTOR_TRANSDUCER	(*)	Yes	-	Immediate
43	fn_PROBING_CONFIGURATION	(see ODM)	No	Wait	
44	fn_SPINDLE_ZERO_OFFSET	deg	No	Immediate	Immediate
45	fn_SPINDLE_WITH_RAMPS	(*)	Yes	-	Immediate
46	fn_SPINDLE_CSS_MASTER	Axis Id	No	Immediate	Immediate
47	fn_SPINDLE_REV_TIME_GEAR1	sec	No	Immediate	Immediate
48	fn_SPINDLE_REV_TIME_GEAR2	sec	No	Immediate	Immediate
49	fn_SPINDLE_REV_TIME_GEAR3	sec	No	Immediate	Immediate
50	fn_SPINDLE_REV_TIME_GEAR4	sec	No	Immediate	Immediate
51	fn_SPIN_POSERR_STANDSTILL	deg	No	Wait	Immediate
52	fn_SPINDLE_SPEED_GEAR1	RPM	No	Wait	Immediate
53	fn_SPINDLE_SPEED_GEAR2	RPM	No	Wait	Immediate
54	fn_SPINDLE_SPEED_GEAR3	RPM	No	Wait	Immediate
55	fn_SPINDLE_SPEED_GEAR4	RPM	No	Wait	Immediate
56	fn_SPINDLE_VOLTAGE_GEAR1	Volt	No	Wait	Immediate
57	fn_SPINDLE_VOLTAGE_GEAR2	Volt	No	Wait	Immediate
58	fn_SPINDLE_VOLTAGE_GEAR3	Volt	No	Wait	Immediate
59	fn_SPINDLE_VOLTAGE_GEAR4	Volt	No	Wait	Immediate
60	fn_SPINDLE_GAIN_GEAR1	IPM / mil	No	Wait	Immediate
61	fn_SPINDLE_GAIN_GEAR2	IPM / mil	No	Wait	Immediate
62	fn_SPINDLE_GAIN_GEAR3	IPM / mil	No	Wait	Immediate
63	fn_SPINDLE_GAIN_GEAR4	IPM / mil	No	Wait	Immediate
64	fn_OFFSET_BETWEEN_MARKERS	impulses	No	Wait	Wait
65	fn_SKEW_ERROR_MIN	M.U.	No	Wait	Wait
66	fn_SKEW_ERROR_MAX	M.U.	No	Wait	Wait
67	fn_SKEW_GAIN	(*)	No	Wait	Wait
68	fn_DRIVE_STATUSW	(*)	Yes	-	Immediate
69	fn_OS3_DRIVE_WARNINGS	(*)	Yes	-	Wait
70	fn_AXIS_IOSTAT	(*)	No	Wait	Wait
71	fn_ACT_MICRO_MARKER_DIST	M.U.	Yes	-	Immediate
72	fn_MAX_MICRO_MARKER_DIST	M.U.	No	Wait	Immediate

73	fn_PROCESS_NUMBER	(*)	Yes	-	Immediate
74	fn_AXIS_SHARING	(*)	Yes	-	Immediate
75	fn_TRANSDUCER_CONFIG	(*)	No	Wait	Immediate
76	fn_ACTIVE_GEAR_NUMBER	#GEAR (0-4)	Yes	-	Immediate
77	fn_ZEROSHIFT1	M.U.	No	No Wait	Immediate
78	fn_ZEROSHIFT2	M.U.	No	No Wait	Immediate
79	fn_AXIS_BACKLASH_TIME	msec	No	Wait	Immediate
80	fn_TRANSDUCER_ID	(*)	Yes	-	Immediate
81	fn_CONVERTER_ID	(*)	Yes	-	Immediate
82	fn_SETRESRIF	(*)	No	Wait	Immediate
83	fn_RAPID_TIME_MIN_RAMP	msec	No	Immediate	Immediate
84	fn_WORKING_TIME_MIN_RAMP	msec	No	Immediate	Immediate
85	fn_RAPID_DECELERATION	M.U. / sec ²	No	Immediate	Immediate
86	fn_MANUAL_DECELERATION	M.U. / sec ²	No	Immediate	Immediate
87	fn_RAMP_MODALITY	(*)	No	Immediate	Immediate
88	fn_GET_MASTER_SLAVE	(*)	Yes	-	Immediate
89	fn_AXIS_INTERFACE_TYPE	(*)	Yes	-	Immediate
90	fn_BW_ENABLE	sec	No	Wait	Immediate
91	fn_BW_TIME	(*)	No	Wait	Immediate
92	fn_EDGE_LEAP_FEED	M.U. / min	No	Immediate	Immediate
93	fn_GET_LAST_ERROR	(*)	Yes	-	Immediate
94	fn_SPINDLE_SS_GAIN_GEAR1	IPM / mil	No	Wait	Immediate
95	fn_SPINDLE_SS_GAIN_GEAR2	IPM / mil	No	Wait	Immediate
96	fn_SPINDLE_SS_GAIN_GEAR3	IPM / mil	No	Wait	Immediate
97	fn_SPINDLE_SS_GAIN_GEAR4	IPM / mil	No	Wait	Immediate
98	fn_SPIN_ORIENT_SPEED_GEAR1	RPM	No	Immediate	Immediate
99	fn_SPIN_ORIENT_SPEED_GEAR2	RPM	No	Immediate	Immediate
100	fn_SPIN_ORIENT_SPEED_GEAR3	RPM	No	Immediate	Immediate
101	fn_SPIN_ORIENT_SPEED_GEAR4	RPM	No	Immediate	Immediate
102	fn_SPIN_ORIENT_ACCEL_GEAR1	rev / sec ²	No	Immediate	Immediate
103	fn_SPIN_ORIENT_ACCEL_GEAR2	rev / sec ²	No	Immediate	Immediate
104	fn_SPIN_ORIENT_ACCEL_GEAR3	rev / sec ²	No	Immediate	Immediate
105	fn_SPIN_ORIENT_ACCEL_GEAR4	rev / sec ²	No	Immediate	Immediate
106	fn_SPIN_STOP_THRESH_GEAR1	RPM	No	Immediate	Immediate
107	fn_SPIN_STOP_THRESH_GEAR2	RPM	No	Immediate	Immediate
108	fn_SPIN_STOP_THRESH_GEAR3	RPM	No	Immediate	Immediate
109	fn_SPIN_STOP_THRESH_GEAR4	RPM	No	Immediate	Immediate
110	fn_SPIN_IN_POS_BAND	deg	No	Wait	Immediate
111	fn_SPIN_IN_POS_WAIT	sec	No	Wait	Immediate
112	fn_SPIN_IN_POS_WINDOW	sec	No	Wait	Immediate
113	fn_ENABLE_SERVO_MODE	(*)	No	Wait	Immediate
114	fn_SKEW_RECOVERY_VEL	M.U. / min	No	Wait	Immediate
115	fn_DRIVE_CONTROLW	(*)	No	Wait	Wait
116	fn_MIN_AX_VELOCITY	M.U. / min	No	Immediate	Immediate
117	fn_MIN_AX_POSITION	M.U.	No	Immediate	Immediate
118	fn_DEN_MEC_PITCH	M.U.	No	Wait	Immediate
119	fn_ZEROSHIFT1_SETUP	M.U.	No	Immediate	-
120	fn_ZEROSHIFT2_SETUP	M.U.	No	Immediate	-

121	fn_RAPID_DJERK	M.U. / sec ³	No	Immediate	Immediate
122	fn_WORKING_DJERK	M.U. / sec ³	No	Immediate	Immediate
123	fn_RAPID_TIME_MIN_DRAMP	Msec	No	Immediate	Immediate
124	fn_WORKING_TIME_MIN_DRAMP	Msec	No	Immediate	Immediate
125	fn_REFERENCE_OFFSET	Volt	No	Immediate	Immediate
126	fn_EMERG_DECELERATION	M.U. / sec ²	No	Immediate	Immediate
127	fn_UPPER_SW_OVERT_TTIP	M.U.	No	Immediate	Immediate
128	fn_LOWER_SW_OVERT_TTIP	M.U.	No	Immediate	Immediate
129	fn_DRIVER_SUBADDRESS	Index	Si	-	Immediate
130	Fn_INTERP_POINT	M.U.	No	Immediate	-
131	fn_LEAP_ACCE	M.U. / sec ²	No	Immediate	Immediate

M.U. stands for “Measuring Unit” and, depending on axis characterisation, it may mean millimetres, inches or degrees. The symbol (*) indicates the presence of special notes given below.

Parameter fx_AXISTYPE is a mask with bits indicating the main characteristics of an axis (for a better understanding, it is advisable to convert it to hexadecimal whole format). The meaning of the bits is given in the following table:

Bit	Val Hex	Mnemonic	Meaning
0	0x0001	AX_LINEAR	Linear axis
1	0x0002	AX_PLA	Axis of logic
2	0x0004	AX_ROTARY	Revolving axis
3	0x0008		reserved
4	0x0010	AX_SPINDNOTRASD	Spindle without transducer
5	0x0020	AX_SPINDRTRASD	Spindle with transducer
6	0x0040	AX_DIAMETRAL	Diametrical axis
7	0x0080		reserved
8	0x0100	AX_VIRTUAL	Virtual axis
9	0x0200	AX_OPTICAL	Axis with optical line
10	0x0400		reserved
11	0x0800	AX_SPINDRAMP	Spindle with ramp
12	0x1000	AX_GANTRY_MASTER	Axis Gantry Master
13	0x2000	AX_GANTRY_SLAVE	Axis Gantry Slave
14	0x4000	AX_ROLLOVER	Rollover axis
15	0x8000	AX_DIGITAL	Digital axis

Parameter fn_CLOCKRAT is a whole number indicating the ratio between the frequency of the clock of the associated interpolator and the frequency of the axis sampling clock.

Parameter fn_DIGITAL_AXIS may have the following values: 0 (NO), 1 (YES).

Parameter fn_LINEAR_OPTICAL_ENCODER may have the following values: 0 (NO), 1 (YES). For HOMING purposes, process axis flags are read at the start of each interpolation, for logic axes they are read only during the OPEN phase. For virtual axes, this function is ignored.

Parameters *fn_CHANNEL_A_POLARITY_INV*, *fn_CHANNEL_B_POLARITY_INV*, *fn_CHANNEL_Z_POLARITY_INV* and *fn_DIRECTION_COUNT* may have the following values: 0 (NO), 1 (YES). It is advisable to disable the axis completely before inverting the polarity of the channels or the count direction of the encoder. This function is ignored for axes with no transducer associated, virtual axes, digital axes.

Parameter *fn_MARKER_DETECTION* may have the following values: 0 (LEVEL), 1 (EDGE). It is not advisable to change marker edge type during the HOMING stage. This function is ignored for: axes with no transducer associated, virtual axes, digital axes.

For parameters *fn_RAPID_TRAVERSE_FEED*, *fn_RAPID_ACCELERATION*, *fn_MANUAL_FEED*, *fn_MANUAL_ACCELERATION*, *fn_RAPID_JERK* and *fn_WORKING_JERK*, for purposes of the movement, the values are considered at the start of the next interpolation. No range check is made.

For parameters *fn_ELECTRICAL_PITCH* and *fn_MECHANICAL_PITCH* it is advisable to disable the axis completely before calling this function. All the parameters associated with the electrical and/or mechanical pitch are recalculated. The function is ignored for: axes with no transducer associated, virtual axes. For split axes, a call to the Master affects the Slave too.

Before changing the parameter *fn_ROLLOVER_PITCH* it is advisable to disable the axis completely before calling the function.

A change in *fn_RAPID_TRAVERSE_VOLTAGE* and *fn_MAXIMUM_FEED* parameters may affect the Servo Error variable algorithm. This function is ignored for: virtual axes, digital axes. It is advisable to disable the axis completely before calling the function.

Parameter *fn_HOME_POSITION_FEED* parameter is taken into account at the start of the next HOMING stage. For split axes, the speed of realignment with the slave is also updated by a call to the master; in this case, the write function becomes wait type. No range check is made.

Parameters *fn_NULL_OFFSET_VALUE* and *fn_HOME_POSITION_VALUE* parameters are actualised at the end of the next reset. It is not advisable to call this function during the HOMING stage. For diametric axes, the write value is halved. No range check is made.

Parameter *fn_HOMING_DIRECTION* may have the following values: 0 (POSITIVE), 1 (NEGATIVE). The value is taken into account at the start of the next HOMING stage. For virtual axes this function is ignored.

Parameter *fn_PERCENT_OF_VFF* is taken into account at the start of the next HOMING stage. For virtual axes this function is ignored.

Parameters *fn_UPPER_SW_OVERTRAVEL* and *fn_LOWER_SW_OVERTRAVEL* are taken into account at the start of the next interpolation. No range check is made. Writing either parameter to zero disables the control of working limits. Writing any value other than zero on either parameter activates the working limit control function. For diametric axes, the value written is halved. For virtual axes this function is ignored.

A change to the *fn_SERVO_LOOP_GAIN* parameter may affect the Servo Error variable algorithm. For virtual axes this function is ignored.

Parameter *fn_STAND_STILL_LOOP_GAIN* is taken into account at the end of the interpolation. For virtual axes this function is ignored.

Parameter *fn_POS_ERROR_STAND_STILL* is taken into account when the axis stops. For virtual axes the function is ignored.

Parameter *fn_POS_ERROR_WITH_VFF* is actualised at the beginning of the next VFF motion.

Parameter *fn_POS_ERROR_WITHOUT_VFF* is actualized at the beginning of the next motion without VFF.

Parameter *fn_IN_POSITION_BAND* is actualized at the end of the interpolation.

Parameters *fn_IN_POSITION_WAIT* and *fn_IN_POSITION_WINDOW* are actualised at the end of the interpolation. No range check is made.

Parameter *fn_AXIS_BACKLASH* is actualised if the axis has been referred to zero. A change to this parameter affects the "Soft" algorithm for the application of mechanical backlash.

No range check is made when the *fn_DEAD_ZONE* parameter is changed.

Parameter *fn_HOMING_TYPE*. The value is taken into account at the beginning of the following HOMING.

No range check is made when the *fn_ADDITIONAL_SERVICE* parameter is changed.

Parameter *fn_MOTOR_TRANSDUCER* may have the following values: 0 (NO) , 1 (YES)

No range check is made when the *fn_PROBING_CONFIGURATION* parameter is changed. The field only concerns the programming of the probing mode for digital drives with D.S.I. interface.

Parameter *fn_OFFSET_FOR_SPINDLE_ORIENT* is taken into account at the start of the spindle orientation stage. No range check is made.

Parameter *fn_SPINDLE_WITH_RAMPS* may have the following values 0 (NO), 1 (YES) according to the active range.

Reversal time is set in parameters *fn_SPINDLE_REVERSAL_TIME_GEAR1/2/3/4*. If the value written is other than zero, the acceleration/deceleration ramp is activated according to the value written; if the value written is zero, the transition from one speed to another is performed with a step. The presence or absence of a ramp, depending on the active range, is denoted by bit 11 "Spindle with ramp" of parameter AXIS TYPE and parameter SPINDLE WITH RAMPS.

For parameters *fn_SPINDLE_SPEED_GEAR1/2/3/4*, *fn_SPINDLE_VOLTAGE_GEAR1/2/3/4* and *fn_SPINDLE_GAIN_GEAR1/2/3/4* function is ignored for virtual and digital axes. It is advisable to disable the spindle completely before calling this function.

For parameters *fn_OFFSET_BETWEEN_MARKERS*, *fn_SKEW_ERROR_MIN* and *fn_SKEW_ERROR_MAX* the function is ignored for virtual axes. It must be used solely on the Master Split axis.

For parameter *fn_SKEW_GAIN* the function is ignored for virtual axes. It must be used solely on the Master Split axis. No range check is made.

For the *fn_DRIVE_STATUSW* parameter encoding, it may assume the following values:

Bit	Val Hex	Mnemonic	Meaning
0..3	0x000X	ST 0..3	Current code status.
4	0x0010	DR_STOP	Fixed axis
5	0x0020	DR_AXREF	Axis referred
6	0x0040	DR_BRAKE	Brake status (0 means brake released or not present)
7	0x0080	DR_HOME	Homing process underway
8	0x0100	DR_ZERO_SW	Zero micro input status
9	0x0200	DR_AUTOTRG	Execution Cmd Point2Point , StepCurr or StepVel
10	0x0400	DR_FFW_ON	Enabled Feed Forward (speed and acceleration)
11			Reserved
12	0x1000	DR_ALARM	Emergency signal presence
13	0x2000	DR_SERVO	Servo Loop are active
14	0x4000	DR_POWER	There is the power: Vbus OK
15	0x8000	DR_MARKER	Found Marker

OS3 drive status encoding:

ST3	ST2	ST1	ST0	Mnemonic	Status
0	0	0	0	DR_START	DRIVE START
0	0	0	1	DR_DIAG	DIAGNOSTIC
0	0	1	0	DR_DRVOFF	DRIVE_OFF
0	0	1	1	DR_DRVON	DRIVE_ON
0	1	0	0	DR_HOLD	HOLD
0	1	0	1	DR_SHTDWN	SHUT_DOWN
0	1	1	0	DR_SHUTTFIX	SHUT_TFIX
0	1	1	1	DR_SHUTIGBT	SHUT_IGBT
1	0	0	0	DR_HALT	HALT

For further details see "OS3 Drive - Installation manual".

For the *fn_OS3_DRIVE_WARNINGS* parameter encoding, see the following table (for further details see OS3 Drive - Installation Manual):

Value	Name	Meaning
0	DR_W_NONE	No warning active
1	DR_W_HEATSINK	Heat sink overheat Warning
2	DR_W_BUSOVERV	Bus Overvoltage Warning
4	DR_W_BUSUNDERV	Bus Undervoltage Warning
8	DR_W_RAPIDHALT	Rapid Halt Warning

For the *fn_AXIS_IOSTAT* parameter encoding, please refer to the following tables:

Possible values read by an OS3 drive

Bit	Name	Meaning
0	DR_GPI2	General Purpose Input 2
1	DR_GPI1	General Purpose Input 1
2	DR_MICROINP	Input Micro of zero

3	DR_RHALTINP	Input of Rapid Halt
4	DR_GP03	General Purpose Output 3
5	DR_GP02	General Purpose Output2
6	DR_GP01	General Purpose Output 1
7	DR_DRDYOUT	Output of Drive Ready
8	DR_BRAKEOUT	Output Motor Brake
9..15		Not used

(for further details see OS3 Drive - Installation Manual)

Possible values written on Bridge device:

Bit	Name	Meaning
0	BR_GPI1	General Purpose Output 1
1	BR_GPI2	General Purpose Output 2
2	BR_GPI3	General Purpose Output 3

Note: To write the outputs, an analog axis with transducer or converter mapped on the bridge has to be declared and the axis ID must be used as function input.

Parameter *fn_MAXIMUM_MICRO_MARKER_DIST* taken in consideration at the beginning of homing procedure.

Parameter *fn_PROCESS_NUMBER* is 0 in the case of a logic axis, it is -1 if the axis has been parked, otherwise it is a value between 1 and 24 reflecting the applicable process.

For coordinated axes, the *fn_AXIS_SHARING* parameter may assume the following values: 0 (not shared), 1 (axis shared by process and logic); for spindle axes: 0 (spindle not associated with any process), 1 (spindle shared by several processes), 2 (spindle exclusive to a process).

Parameter *fn_ACTIVE_GEAR_NUMBER* returns values from 1 to 4 for spindle axes and 0 for non-spindle axes.

Values *fn_ZEROSHIFT1* and *fn_ZEROSHIFT2* are interpreted as absolute values to be forced in the zero shift registers of the servo processor modulus. The offsets are actualised regardless of whether or not an axis has already been referred. WinPLUS is able to move the selected axis by means of this function. The current axis position register and the request position registers will not change. Zero shift offsets may be used to compensate for axis errors (e.g., temperature) or to bring about a physical movement of the origin. An offset can be removed through the *AX_ResetPosition* function. It is not possible to call again the function during the execution to write another zero shift and it is not possible to give the following commands: *AX_ResetPosition*, *AX_Enable*, *AX_Disable*.

The offset value contained in the zero shift register can be initialized through the function *AX_ZEROSHIFT1_SETUP* and *AX_ZEROSHIFT2_SETUP*, passing as parameter the initialization value; this value will be bound to the zero shift subtracting the axis position. Passing to 0.0 value, the zero shift contribution is null as happened with the function *AX_ResetPosition*.

For parameters *fn_TRANSDUCER_ID* and *fn_CONVERTER_ID* see the notes in the preface of library manual.

Parameter *fn_SETRESRIF* may take values 0 (RESRIF) and 1 (SETRIF). It can only be written; to determine the axis referred condition, use function AX_ReadStatus. The condition of logic reference is returned in the case of reading (bit 6 of status word Status 1). This condition indicates that the axis has been referred manually or via zero setting procedure (read bit 6 of status word Status2 to know the condition of physical reference). The logic reference of the axis is made with writing of value 1 (SETRIF) (manually procedure), instead zero setting procedure and all axis physical offset (also Null Offset and Home Position) are deleted with writing of value 0 (RESRIF) (see the function AX_ResetPosition with parameter Mode=4). Status1 and Status2 words can be read using the function AX_ReadStatus.

Parameter *fn_RAMP_MODALITY* indicates the acceleration ramp modality.

Parameter *fn_GET_MASTER_SLAVE_ID* indicates the ID of the complementary axis in the case of Gantry axes.

Parameter *fn_AXIS_INTERFACE_TYPE* indicates one of the following types of axis interface:

Value	Name	Meaning
0	ANALOG_INTERF	Analog interface, used by analogic axes and axes on Os-Wire Bridge
1	SERCOS_INTERF	SERCOS interface
2	MECHAT_INTERFACE	Mechatrolink interface
3	OSWIRE_INTERF	OS-Wire interface, used by digital axes with OS3 drive
4	CANOPE_INTERF	CANOpen interface, used by digital axes with VFD drive
5	CANA10_INTERF	For analog axes with A10 CANOpen analog modulus
6	CANP24_INTERF	For analog axes with P24 CANOpen analog modulus
8	ETCSOE_INTERF	For digital axes with drive having a SERCOS Over EtherCAT profile
9	ETCNLG_INTERF	For analog axes with EtherCAT PE E-501 analog modulus

Parameter *fn_BW_ENABLE* may take the following values: 0 (NO), 1 (YES).

Parameter *fn_BW_TIME* is taken into account when the axis is enabled.

fn_GET_LAST_ERROR reads the real-time error code of the axis

For parameters *fn_SPINDLE_SS_GAIN_GEAR1/2/3/4* the function becomes wait type if the change is made on the active range.

Parameters *fn_SPIN_ORIENT_SPEED_GEAR1/2/3/4* and *fn_SPIN_ORIENT_ACCELERATION_GEAR1/2/3/4* are considered at the start of the spindle orientation stage. No range check is performed.

Parameter *fn_SPIN_STOP_THRES_GEAR1/2/3/4* is taken into consideration at the start of the boring cycle. No range check is performed.

Parameter *fn_ENABLE_SERVO_MODE* indicates the servo mode applied to the axis by the enable command (function AX_ENABLE with resource 1). The admissible values are those associated with the Primary Operating Mode and the Secondary Operating Mode of the axis (see function AX_ReadStatus, status word 2, masks AX_MODE1MSK and AX_MODE2MSK).

Parameter *fn_SKEW_RECOVERY_VEL* indicates the speed of the Slave Split axis during the realignment stage. When this parameter is written its value must not exceed manual feed rate, or an error will occur.



The modification of some parameters is dangerous and it can determine instability of the axis.

Return values:

The following table lists possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors.
0x00200001	Insufficient memory to elaborate.
0x00200003	Too many requests to axes board.
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Call to virtual axis.
0x00200031	Call to split slave axis.
0x00200080	Param not valid.
0x00200081	Id axes not congruent.
0x00200082	Error during calculation of position gain.
0x00200083	Error during calculation of IN POSITION BAND.
0x00200084	Value outside range. Error of conversion in internal units.
0x00200085	ELECTRICAL PITCH wrong
0x00200086	Axis without transducer.
0x00200087	GEAR number wrong.
0x00200088	Digital axis not available.
0x0020008d	Axis without converter

Example:

```
Ret : dword;
Ret := AX_WriteParameter (1, fn MANUAL FEED, 1000.0);
```

Changes the speed of manual movement at 1000 mm/min of the axis with ID=1.

See also:

[AX_ReadParameter](#), [AX_ReadStatus](#)

7.12 AX_ChangeServoMode

The AX_ChangeServoMode function change servo loop mode.

Syntax:

```
ret_code := AX_ChangeServoMode (AxisId, ServoMode) ;
```

Input parameters:

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>ServoMode</i> (DWORD)	Servo loop mode

Execution mode:

NoWait

Use:

This function can be used to change the axis servo loop mode. The values can be:

ServoMode =	0x00200000	Feed rate servo loop mode, no position control.
	0x00300000	Servo mode open loop position. Feed rate=0 and following error is controlled.
	0x00310000	Servo mode closed loop position Feed rate=0 and following error is controlled and recovered.

The servo loop mode set is active until the next motion, when the interpolator sets the default one. Error returns if there are digital axes or if the axis is in test mode.



In order to have the maximum flexibility, the consistency between value set and the axis is not checked to avoid wrong motions of the axes, the user has to set the right values.

AX_ReadStatus ensures the servo loop mode change is done.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200003	Too many requests form the axis board
0x00200004	Wrong AxisId
0x00200006	Axis not configured
0x0020000F	Invalid request
0x00200084	Invalid ServoMode

Example:

```
Ret : dword;
Ret := AX_ChangeServoMode (1, AX_MODE1_POS or AX_MODE2_SPC);
```

Axis ID=1 changes the servo loop mode in StandStill Closed Loop.

See also:

[AX_ReadStatus](#)

7.13 AX_CheckHardwareOvertravel

Function AX_CheckHardwareOvertravel manages the hardware limit switches of an axis.

Syntax:

```
ret_code := AX_CheckHardwareOvertravel (ExecMode, AxisId, MicroPos, MicroNeg, Mode) ;
```

Input variables:

<i>ExecMode</i> (INT)	Execution mode [MODE_WAIT, MODE_NOWAIT]
<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>MicroPos</i> (BOOL)	Positive Physical Input of micro switch
<i>MicroNeg</i> (BOOL)	Negative physical input of the micro switch
<i>Mode</i> (BOOL)	Deceleration modality

Execution mode:

Wait, NoWait

Use:

The mode parameter value should be FALSE to apply the deceleration ramp when the axis reaches the limit switch; it should be TRUE to stop the axis without a ramp when it reaches a limit switch (i.e., the reference value is set to zero).

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors.
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x00200016	(axis enable / disable) command already active.
0x00200017	(axis enable / disable) command activation conditions not present.

Example:

```
Ret : dword;
MicroPos : bool;
MicroNeg : bool;
Ret := AX_CheckHardwareOvertravel (MODE_WAIT, 1, MicroPos, MicroNeg, FALSE);
```

For ID=1 axis, hardware over travel micros are managed with ramp in wait mode.

7.14 AX_SetHardwareOvertravel

Function AX_SetHardwareOvertravel sets the status of hardware over travel limit switches.

Syntax:

```
ret_code := AX_SetHardwareOvertravel (AxisId, Direction, OvertState) ;
```

Input parameters:

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Direction</i> (INT)	Limit switch direction
<i>OvertState</i> (INT)	Status of over travel micro

Execution mode:

NoWait

Use:

Any coordinated axis can be equipped with two hardware over travel limit switches indicating axis over travel. When an axis hits the switch, you use this function is used to advise the system of this error status by means of parameter *OvertState* = AX_OVERTRAVEL_ACTIVED (0). The *Direction* parameter defines which limit switch has been pressed, either the positive one AX_OVERTRAVEL_POS (1) or the negative one AX_OVERTRAVEL_NEG (0).

The system will execute following action:

- › stops the interpolation for all the process axes
- › Send a message "AXIS HW OVERTRAVEL LIMIT REACHED" to the screen.

The process is now in stand-by and will only allow the axis on the over travel limit switch to be moved in manual mode and in the opposite direction (away from the over travel limit switch). The homing movement of the axis (HOME) is inhibited if either micro switch is pressed. To enable normal operation of the axis, you have to use the same function with parameter *OvertState* = AX_OVERTRAVEL_EXIT (1) and parameter *Direction* to release the micro switch released. From this moment, it will possible to execute motions again and in all operating modes and in both directions.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	AxisId wrong.
0x00200006	Axis not configured.

Example:

```
Ret : dword;
Ret:=AX_SetHardwareOvertravel (2,AX_OVERTRAVEL_POS,AX_OVERTRAVEL_ACTIVED) ;
```

For the axis with ID=2 hardware over travel condition in the positive direction is notified.

7.15 AX_Disable

Function AX_Disable allows the deactivation of serving loops on a real axis

Syntax:

```
ret_code := AX_Disable (ExecMode, AxisId, Mode) ;
```

Type: No Wait

Input variables:

<i>ExecMode</i> (INT)	Execution Mode [MODE_WAIT, MODE_NOWAIT]
<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Mode</i> (INT)	resource to be disabled [1..2]

Execution mode:

Wait, NoWait

Use:

This function makes it possible to disable the control processes of an axis. The admissible operations are listed in the following table by type of resource.

Mode	Description
1 (axis resource) ENAB_AXIS	For all types of axis, the servo management routine is disabled. In the case of coordinated and logic axes, the position error is not controlled. In the case of analogic axes, the D/A converter is set to 0 volt and the BROKEN WIRE control is deactivated. For digital axis, see resource 2.
2 (drive resource) ENAB_DRIVE	For digital axes, disables the associated drive: <ul style="list-style-type: none"> ➢ Axes with D.S.I.: sets the drive to DRIVE ON=0 and ENABLE DRIVE=0 ➢ Axes with OS3: transmits drive disable command (DRIVE OFF). ➢ Mechatrolink axes: sends the SV_OFF drive disable command.

In MODE_WAIT execution mode, the function waits until the resource has been disabled.

In MODE_NOWAIT execution mode, the function starts the disable procedure and does not wait for the end of the process; in order to determine whether the disable procedure has been completed, make sure that the corresponding command pending bit (bit 11 - AX_CMD_DISA) is 0. As long as the bit is 1 the disable process is underway and during this stage it is not possible to recall this function or one of the following functions: AX_Enable, AX_ResetPosition, ZeroShifts write. To read the commands pending on axis status, call function AX_ReadPendingCommands.

For further details about the disable procedure see application manual.



This function manages an axis in its inactive state, during which the drive/motor assembly control loops are no longer active. In the case of vertical axes, check functionality of mechanical brake.

In the case of Gantry axes, the function can be called only for the master axis.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors.
0x00200004	AxisId wrong
0x00200006	Axis not configured
0x0020000D	Invalid resource for analog axes.
0x0020000F	ExecMode or Mode out of range. Call to virtual axis.
0x00200016	Command already active.
0x00200017	Command activation conditions not present.
0x00200031	Call to split slave axis.
0x00200070	Communication with digital axis timeout.
0x00200071	Communication with digital axis channel busy.
0x00200072	Communication with digital axis channel deactivated due to severe error.
0x00200073	Stage of communication with digital axis non-conforming.
0x0018xxxx	Errors returned from Os-Wire environment.

Example:

```
Ret : dword;
Ret := AX_Disable (MODE_NOWAIT, 1, ENAB_AXIS);
```

The axis with ID=1 is disabled in NoWait mode.

See also:

[AX_Enable](#), [AX_ReadStatus](#), Application Manual

7.16 AX_Enable

Function AX_Enable permits the activation of serving loops on a real axis.

Syntax:

```
ret_code := AX_Enable (ExecMode, AxisId, Mode) ;
```

Input variables:

<i>ExecMode</i> (INT)	Execution mode [MODE_WAIT, MODE_NOWAIT]
<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Mode</i> (INT)	Resource to be enabled [1..2]

Execution mode:

Wait, NoWait

Use:

The function permits the activation of serving loops on a real axis. The following table indicates the admissible operations by type of resource.

Resource	Description
1 axis resource ENAB_AXIS	Activation of default servo routine. For all coordinated and logic axes, position error control is activated, in the case of a spindle axis, speed management is activated. For analog axes, the position loop is closed (possible deviations from the required position are corrected, by sending a voltage signal to the D/A converter of the axis). BROKEN WIRE control (if enabled) is activated on the transducer; the user can set a delay time for this detection with parameter 91 AX_WRITE Parameter function. For digital axes, see resource 2.
2 drive resource ENAB_DRIVE	For digital axes, enables and closes the position loops on the associated drive: ➢ Axes with D.S.I.: sets the drive to DRIVE ON=1 and ENABLE DRIVE=1 ➢ Axes with OS3: transmits drive enable command (DRIVE ON). ➢ Mechatrolink axes: sends the SV_ON drive enable command.

In MODE_WAIT execution mode, the function waits until the resource has been enabled.

In MODE_NOWAIT execution mode, the function starts the enable procedure and does not wait for the end of the process; in order to determine whether the enable procedure has been completed, make sure that the corresponding command pending bit (bit 10 - AX_CMD_ENAB) is 0. As long as the bit is 1 the enable process is underway and during this stage it is not possible to recall this function or one of the following functions: AX_Disable, AX_ResetPosition, ZeroShifts write.

To read the commands pending on axis status, call function AX_ReadPendingCommands.

For further details about the enable procedure, see the application manual.



This function manages an axis in its active state, during which the drive/motor assembly control loops are active. Before calling this function, make sure that the moving parts connected to the motor are in safety conditions.



In the case of Gantry axes, the function can be called only for the master axis; in the event of a misalignment between the master axis and the slave axis, you can use the AX_ResetPosition (Mode=2) function to realign the axes.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors.
0x00200004	AxisId wrong
0x00200006	Axis not configured
0x0020000D	Invalid resource for analog axes.
0x0020000F	ExecMode or Mode out of range. Call to virtual axis.
0x00200016	Command already active.
0x00200017	Command activation conditions not present.
0x00200031	Call to split slave axis.
0x00200062	Axis moving.
0x00200070	Communication with digital axis timeout.
0x00200071	Communication with digital axis channel busy.
0x00200072	Communication with digital axis channel deactivated due to severe error.
0x00200073	Stage of communication with digital axis non-conforming.
0x0018xxxx	Errors returned from Os-Wire environment.

Example:

```
Ret : dword;
Ret := AX_Enable (MODE_WAIT, 1, ENAB_AXIS);
```

The axis with ID=1 is enabled in wait mode.

See also:

[AX_Disable](#), [AX_ReadStatus](#), [AX_ReadPendingCommands](#), Application Manual.

7.17 AX_Reset

The function AX_Reset is used to abort a command on the axis.

Syntax:

```
ret_code := AX_Reset (AxisId) ;
```

Input parameter:

AxisId (INT) Axis identifier [1..64]

Execution mode:

NoWait

Use:

This function is used for the axes of logic. It is used to abort the command on the marker and the probe signal, the interpolator and hence the movement are not reset. At the end of the function it is not certain whether the command has been really aborted, the operation may request several servo cycles (depending on type of command to be aborted); other functions have to be used (e.g., AX_ReadStatus) to determine the actual status of the axis.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Call to virtual axis.

Example:

```
ret: dword;  
ret := AX_Reset (1);
```

Aborts axis controls with ID=1.

7.18 AX_ReadStatus

The function AX_ReadStatus reads the state of the axis.

Syntax:

```
ret_code := AX_ReadStatus (AxisId, Status1, Status2);
```

Input parameter:

AxisId (INT) axis identifier [1..64]

Output parameters:

Status1 (DWORD) status dword 1

Status2 (DWORD) status dword 2

Execution mode:

Immediate.

Use:

This function makes it possible to determine the status of an axis through two DWORDs seen as bits. The first Dword defines an axis “configuration” status, while the second Dword defines a status that may vary dynamically during machining operations.

Below the description of the *Status1* variable:

Bit	Val Hex	Mnemonic	Meaning
0	0x00000001	AX_DIS_NEGLOP	Negative software limits disabled. It is set to 1 when the check on the negative operating limit is disabled.
1	0x00000002	AX_DIS_POSLOP	Positive software limits disabled. It is set to 1 when the check on the positive operating limit is disabled.
2..3	-		Reserved
4	0x00000010	AX_ON_NEGOVERT	Negative over travel micro active. It is set to 1 when the mechanical micro switch is pressed.
5	0x00000020	AX_ON_POSOVERT	Positive over travel micro active. It is set to 1 when the mechanical micro switch is pressed.
6	0x00000040	AX_HOME_DONE	Axis referred. It is set to 1 when the homing procedure has been executed, or when the axis has been forcedly referred by the PLC or by the Process Controller. It is set to 0 when an axis has never been referred.
7	0x00000080	AX_LOP_ENABLE	Operating limits have been defined.
8	0x00000100	AX_HOMING_RUN	Homing underway. The value is 1 throughout the execution of the axis homing process.
9..26	-		Reserved
27	0x08000000	AX_IS_SHARED	Shared axis. It is set to 1 when an axis is shared with a PLC process (a sharing function has been executed); it goes back to 0 with the axis release

Bit	Val Hex	Mnemonic	Meaning
			command.
28..31	-		Reserved

Status2 variable description:

Bit	Val Hex	Mnemonic	Meaning
0	0x00000001	AX_ERROR_STS	Axis in error. It means that an operation on the axis has given rise to an error (legible with AX_ReadParameter). An error status occurs whenever the axis generates an emergency condition. The bit is set back to 0 following a call to AX_Enable with resource 1.
1	0x00000002	AX_TEST_STS	Axis in test mode.
2	0x00000004	AX_TOL_STS	Axis in tolerance range. It is set to 1 when the axis is in position tolerance and it is set to 0 when the entry into tolerance process is underway.
3	0x00000008	AX_MOVE_STS	Axis interpolation. This is set to 1 when the axis is moved by the process or logic interpolator.
4..5	-		Reserved
6	0x00000040	AX_HOMED_STS	Axis homed. It is set to 1 when the homing process has been completed successfully.
7	0x00000080	AX_MICROSW_STS	Digital drive micro switch status. It is set to 1 when the micro switch connected to a digital drive is pressed.
8..13	-		Reserved
14	0x00004000	AX_HOME_DIR_STS	Home micro switch search direction. It is set to 1 when the home micro switch search direction is negative.
15	-		Reserved
16..19	0x000X0000	AX_MODE2MSK	Secondary operating mode.
20..23	0x00X00000	AX_MODE1MSK	Primary operating mode
24	0x01000000	AX_DRVREADY1	Drive on
25	0x02000000	AX_DRVREADY2	Drive enable
26..31	-		Reserved

Bits from 20 to 23 indicate the primary operating mode that is active on the axis. These 4 bits are seen as follows:

Primary operating mode (hex)	Mnemonic	Meaning
0	AX_MODE1_DISAB	Disabled axis
1		reserved
2	AX_MODE1_VEL	Enabled axis in speed check
3	AX_MODE1_POS	Enabled axis with position loop
4 - F		reserved

Bits from 16 to 19 indicate the secondary operating mode active on the axis, whose meaning depends on the primary mode. It is valid only when the operating mode is 3. These 4 bits are seen as follows:

Secondary operating mode (hex)	Mnemonic	Meaning
0		reserved
1	AX_MODE2_SPC	Closed space loop active (following error checking and recovery).
2	AX_MODE2_VFF	Closed speed loop active (determination of speed + following error checking and recovery).
3 - F		reserved

Bits 24 e 25 (Drive ON and Drive ENABLE) indicate enable status of a digital drive (D.S.I. or OS-Wire) and must be interpreted as a binary code having the values given in the following table:

AX_DRVREADY2	AX_DRVREADY1	Meaning
0	0	Drive not ready for power up
0	1	Drive ready for main power on
1	0	Drive ready and main power applied
1	1	Drive ready to operate

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Call to virtual axis.

Example:

```
Ret : dword;
STAT1: dword;
STAT2: dword;
Ret := AX_ReadStatus (1, STAT1, STAT2);
```

Reads axis ID=1 status.

7.19 AX_SetStatus

Function AX_SetStatus sets a number of axis status flags.

Syntax:

```
ret_code := AX_SetStatus (AxisId, Mask, Status);
```

Input parameters:

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Mask</i> (DWORD)	Status flag selection mask
<i>Status</i> (DWORD)	Value of status flags

Execution mode:

Immediate.

Use:

This function makes it possible to force the condition of several axis status flags. The flags to be modified must be indicated in the *Mask* input variable; the values to be assumed by the flags are defined in *Status*.

Description of variable *Mask*.

Bit	Val Hex	Mnemonic	Meaning
0	0x00000001	AX_DIS_NEGLOP	Negative software limits disabled. It is set to 1 to disable the negative operating limit.
1	0x00000002	AX_DIS_POSLOP	Positive software limits disabled. It is set to 1 to disable the positive operating limit.
2..3	-		Reserved.
4	0x00000010	AX_ON_NEGOVERT	Negative over travel micro switch active. It is set to 1 to indicate that the mechanical micro switch has been pressed.
5	0x00000020	AX_ON_POSOVERT	Positive over travel micro switch active. It is set to 1 to indicate that the mechanical micro switch has been pressed.
6	0x00000040	AX_HOME_DONE	Axis referred. Set it to 1 to indicate that the axis has been referred.
7	0x00000080	AX_LOP_ENABLE	Operating limits defined.
8	0x00000100	AX_HOMING_RUN	Homing underway. It is 1 throughout the execution of the axis homing procedure.
9..26	-		Reserved
27	0x08000000	AX_IS_SHARED	Axis shared. It is set to 1 when the axis is shared with PLC process (a shared function has been executed); it goes back to 0 with the axis release command.
28..31	-		Reserved

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors.
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Call to virtual axis.

Example:

```
Ret : dword;
Stat : dword;
Mask : dword;
Mask := AX_DIS_NEGLOP or AX_DIS_POSLOP;
Stat := 0;
Ret := AX_SetStatus (1, Mask, Stat);
```

For the axis with ID=1 the software limits are enabled.

7.20 AX_ReadPendingCommands

Function AX_ReadPendingCommands makes it possible to determine the status of the inner commands of the axis.

Syntax:

```
ret_code := AX_ReadPendingCommands (AxisId, Commands) ;
```

Input parameter:

AxisId (INT)	Axis identifier [1..64]
--------------	-------------------------

Output parameter:

Commands (DWORD)	Pending commands mask
------------------	-----------------------

Execution mode:

Immediate.

Use:

This function may be used to determine the status of a number of internal axis commands according to the following table:

Commands	Value	Description
AX_CMD_PROBING	0x00000001	Probing underway.
AX_CMD_MARKER	0x00000002	Marker search underway.
AX_CMD_INITPOS	0x00000004	Axis position initialisation underway.
AX_CMD_INTERP	0x00000008	Interpolation underway.
AX_CMD_TOLLER	0x00000010	Entry into tolerance field underway.
0x00000020	0x00000020	Reserved.
0x00000040	0x00000040	Reserved.
AX_CMD_RESETPOS	0x00000080	Execution of function AX_ResetPos underway.
AX_CMD_ZEROSH1	0x00000100	Zeroshift 1 actualisation.
AX_CMD_ZEROSH2	0x00000200	Zeroshift 2 actualisation.
AX_CMD_ENAB	0x00000400	Enable process underway
AX_CMD_DISA	0x00000800	Disable process underway
AX_CMD_RESCMD	0x80000000	Commands reset underway.
Others		Reserved.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors.
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Call to virtual axis.

Example:

```
Ret : dword;  
Commands : dword;  
Ret := AX_ReadPendingCommands (1, Commands);
```

Reads the status of commands for axis with ID 1.

7.21 AX_Filter

Function AX_Filter configures the filters of an axis.

Syntax:

```
ret_code := AX_Filter (AxisId, Command, Filter, Value) ;
```

Input parameters:

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Command</i> (WORD)	Command [0..5]
<i>Filter</i> (WORD)	Filter identifier [1..3]

Input/output parameters:

<i>Value</i> (ARRAY [0..7] OF LREAL)	Written/read parameters of filter
--------------------------------------	-----------------------------------

Execution mode:

Immediate

Use:

This function makes it possible to use axis filters. The filters are used in a wide variety of applications: to regularise a set of values, to process data while eliminating a specific proportion thereof, or for the finite band pre-compensation of control loops. Before it can be used, a filter must be configured and enabled, and hence when it is no longer needed it can be disabled and deleted.

Possible filters:

Filter =	SMOOTH_FILTER	Point average filter
	:C_FILTER	Centripetal force compensation filter
	:ART_FILTER	Art or reversal compensation filter

The commands are listed in the following table:

Command	Description
FILTER_READ	Read filter parameters
FILTER_WRITE	Allocate a new filter or rewrite filter parameters
FILTER_DELETE	Delete filter (must be disabled)
FILTER_ENABLE	Enable filter
FILTER_DISABLE	Disable filter
FILTER_STATUS	Filter status query; one of the following values is returned in Value[0]: -1 filter not configured 0 filter inactive 1 filter active

Filter : SMOOTH_FILTER

With this type of filter it is possible to average all the points and speeds sent to the drives over a configurable number of samples. This function improves the dynamic behaviour of an electromechanical system in the case of programming by points mated to low tolerances. By executing a mobile average over the points it is possible to smooth the corners of the polygonal generated by CAD systems. The only filter parameter, written and read in Value[0], is the lower and upper sampling limit (L). The total number of points taken into account in order to determine the average reference applied to the system will be $2*L+1$; thus the point obtained will correspond to the current point averaged with the L points that precede it and the L points that follow it. The maximum admissible value is 32.

Filter : ACC_FILTER

During a circular interpolation, the typical error is a reduction in the size of the measured radius vs. that of the programmed radius. The centripetal acceleration correction filter compensates for the following error in order to obtain a behaviour that is similar to the behaviour obtained through the application of Velocity Feed Forward, but will not make the machine tool too jumpy. The only filter parameter, written and read in Value[0], is the gain for position loop Kv which may also differ from the value set in AMP.

Filter : START_FILTER

This filter makes it possible to take up the “elastic play” that occurs at axis reversal or start, due to the presence of non-linear friction arising from the minimum torsion taking place in the motor. Filter parameters are:

Value[0] : rising constant of first order filter (value between 0 and 1)

Value[1] : falling constant of first order filter (value between 0 and 1)

Value[2] : position jump to be taken up (mm/inches/degrees)

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors.
0x00200001	Not enough memory to execute the operation.
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Command wrong.
0x00200062	Filter enable or write during axis movement.
0x00200065	Filter not configured.
0x00200066	Filter wrong.
0x00200067	Command cannot be executed in the following cases: deletion of a filter when it is active.
0x00200084	One of the parameters in Value is out of range.

Example :

```

Ret : dword;

FltValue : array [0..7] of lreal;

FltValue[0] := 5;                                     (* Average on 11 points *)

Ret := AX_Filter (1, FILTER_WRITE, SMOOTH_FILTER, FltValue);

Ret := AX_Filter (1, FILTER_ENABLE, SMOOTH_FILTER, FltValue);

Sets and enables axis ID=1filter average on points.

```

7.22 AX_EnableProbe

The AX_EnableProbe function enables the storage of the axes positions specified when is generated the probe signal.

Syntax:

```
ret_code := AX_EnableProbe (AxisId1 [, AxisId2, ... , AxisId64]);
```

Input parameter:

<i>AxisId</i> (INT)	Axis identifier [1..64]
---------------------	-------------------------

Execution mode:

NoWait

Use:

After probing cycle, the probe has to be enabled with a new AX_EnableProbe call in order to recognize the following cycle. Distances memorized can be read using AX_ReadProbePosition function. The call of AX_Reset resets the command. Axes involved cannot be virtual.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	AxisId wrong.
0x00200006	AxisId not configured.
0x0020000F	Unaccepted for virtual axis
0x00200016	Instruction already active

Example:

```
Ret : dword;
Ret := AX_EnableProbe (1, 2, 3);
Enables probing on three axes with ID 1, 2 and 3.
```

See also::

[AX_ReadProbePosition](#), [AX_ReadProbeStatus](#), [AX_Reset](#).

7.23 AX_ReadProbePosition

Function AX_RDPB reads probed values of the axes involved in the probing cycle.

Syntax:

```
ret_code := AX_ReadProbePosition (Position, AxisId1 [, AxisId2, ..., AxisId64]);
```

Input parameters:

AxisId (INT) Axis identifier [1..64]

Output parameters:

Position (ARRAY OF LREAL) Array probed values

Execution mode:

NoWait

Use

This function must be called after executing the probe cycle. The probed positions returned by the function AX_ReadProbePosition are values expressed in configured units (mm, inches or degree).

Reading of the probed value DOES NOT take into consideration Horizontal Shift (present in UPA process variable), Vertical Shift (present in UPO process variable) and probe ball radius.

In order to calculate the actual value recorded with respect to the concerned axis' home position, it is necessary to make the following calculation:

value recorded by the logic + probe ball radius - Horizontal Shift

or

value recorded by the logic + probe ball radius - Vertical Shift

Horizontal Shift and *Vertical Shift* values must be used with the relative sign.

Example:

Value read through function	= 109.7464
Probe ball radius	= 0.9801
Horizontal Shift	= 0.2671
Actual value = (109.7464 + 0.9801 - (- 0.2672))	= 110.9937



No check is made on the number of *AxisId* duplications; the execution of the function with these wrong parameters may generate anomalies.

Return values:

ret_code (HEX)	Description
0x00000000	Function performed without errors
0x00200004	AxisId wrong.
0x00200006	AxisId not configured.
0x0020000F	Not admissible for virtual axis.

Example:

```
Ret : dword;
Val : array [1..3] of lreal;
Ret := AX_ReadProbePosition (Val, 1, 2, 3);
```

Reads probed values for axes with ID 1, 2 and 3.

See also:

[AX_EnableProbe](#), [AX_ReadProbeStatus](#).

7.24 AX_ReadProbeStatus

Function AX_ReadProbeStatus determines probe status for the axes specified.

Syntax:

```
ret_code := AX_ReadProbeStatus (Status, AxisId1 [, AxisId2, ..., AxisId64]);
```

Input parameters:

AxisId (INT) Axis identifier [1..64]

Output parameters:

Status (DWORD) Probe status

Execution mode:

Immediate.

Use:

This function may be used to determine the outcome of the probing process after the call to function AX_EnableProbe. The *Status* output variable may take on the following values:

Status	Value	Description
AX_PROBING	0	Probing underway.
AX_PROBE_ANOMALY	1	Anomaly or probe already pressed.
AX_PROBED	2	Probing executed.
AX_PROBE_NOTACTIVE	4	Probing not active yet.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors.
0x00200004	AxisId wrong.
0x00200006	AxisId not configured.
0x0020000F	Not admissible for virtual axis.

Example:

```
Ret : dword;
Status : dword;
Ret := AX_ReadProbeStatus (Status, 1, 2, 3);
```

Reads probe status for the axes with IDs 1, 2, and 3.

See also:

AX_EnableProbe.

7.25 AX_SetTestMode

Function AX_SetTestMode sets the input or the output to test mode for the axes specified.

Syntax:

```
ret_code := AX_SetTestMode (Status, AxisId1 [, AxisId2, ..., AxisId64]);
```

Input parameters:

<i>Status</i> (BOOL)	Input/ output test mode [true, false]
<i>AxisId</i> (INT)	Axis identifier [1..64]

Execution mode:

Immediate

Use:

When an axis is put in test mode (*Status* = true), it will not physically move. The axis is kept in closed loop at the position it was in when the test mode was activated. In the display, the axis positions are updated as if the axis was moving. All CNC functionality can be applied to an axis in test mode. An axis in interpolation state cannot be set to test mode. After returning from test mode (*Status* = false), all registers and flags concerning that axes are restored to the state they were in before the axis was put into test mode. This function can be called only with the axes stationary.



When an axis is in test mode, none of the axis movements set by the part-program are executed. Make sure that calling this function will not cause irregularities in the interpolation with other axes.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors.
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Call to virtual axis.
0x00200062	Moving axis.

Example:

```
Ret : dword;
Ret := AX_SetTestMode (true, 1);
```

7.26 AX_ResetPosition

Function AX_ResetPosition sets the axis position to zero.

Syntax:

```
ret_code := AX_ResetPosition (AxisId, Mode) ;
```

Input parameters:

<i>AxisId</i> (INT)	axis identifier [1..64]
<i>Mode</i> (INT)	mode [0..4]

Execution mode:

NoWait

Use:

Description of possible operations:

Mode AX_RESET_POS (0)

The function sets the axis position to zero; the new value displayed is given by the remainder resulting from the division between the axis position and the mechanic pitch.

This way of taking to zero is used for axes that move always in the same direction (for example a rotary axis that rolls up a thread or a "tapis roulant" linear axis); in this case, if AX_ResetPosition is not used, the screen position is always increased until the maximum position is exceeded (see formula below) generating a system anomaly.

$$\text{maximum position} \quad \text{-----} \quad \begin{matrix} \text{(maximum number of impulses)} \\ \text{* mechanical pitch} \end{matrix}$$

$$\qquad\qquad\qquad \text{electrical pitch}$$

"Maximum number of impulses" is: $(2^{63}) - 1$.

If a Home Position value is present, the point where the axis' position is reset becomes the new Home Position point. If Zeroshift values are active when the AX_ResetPosition function is executed, these values are reset after the axis position is reset. The function is not executed if the axis is in motion or if the axis is locked.

Mode AX_RESET_HOMEPOS (1)

Forces the position of the axis to the Home Position value valid at the moment, by resetting the Zeroshift 1/2 and Nulloffset interior registers.

Mode AX_ALIGN_GANTRY (2)

Brings about the gradual realignment of two gantry axes. This function should be recalled only for the master axis, and the axes must have been referred at least once.

Mode AX_RESET_ZERO_SHIFT (3)

Zeroshift 1 and 2 registers are reset. The effect is that the sum of the two registers is added to the calculation point and therefore this function can be called with the axes enabled. In the case of gantry axes this function must be called only for the master axis, not for the slave, and the Zeroshift registers will be reset for both. If the two split axes are offset relative to one another on account of their having different Zeroshift values, this reset causes them to be immediately realigned.

Mode AX_RESET_HOMING (4)

Sets the axis in «not-referred» mode, i.e., it neutralises the effect of the axis reset procedure. All the physical offsets applied (Null Offset, Home Position, ZeroShifts, geometric corrections and axis backlash) are neutralised, the machine position of the axis is made to correspond to the physical position.



This is a NO WAIT type function. To determine when the function has actually been completed (for example, to determine whether the gantry axes realignment process is over), you have to call the AX_ReadPendingCommands function to read pending commands and test bit 7 (value AX_CMD_RESETPOS 0x00000080): as long as the bit continues to be 1 the function is being executed, when the bit changes to zero the activity is over.

During the execution of the function it is not possible to launch the following commands: AX_ResetPosition, AX_Enable, AX_Disable and ZeroShifts write commands. Moreover, the axis cannot be moving and in Homing stage.

Return values:

The following table lists all the possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200003	Too many requests on the axis board.
0x00200004	AxisId wrong.
0x00200006	Axis not configured.
0x0020000F	Call to virtual axis. Mode not valid.
0x00200016	Command already active
0x00200017	Conditions to activate the command do not exist

Example:

```
ret: dword;
ret := AX_ResetPosition (1, AX_RESET_POS);
```

Executes a position reset in AX_RESET_POS (0) mode for the axis with ID=1.

7.27 AX_SelectGearFoller

Function AX_SelectGearFoller makes it possible to select a range for spindle axes, or to set the type of servo error of an axis.

Syntax:

```
ret_code := AX_SelectGearFoller (AxisId, Mode) ;
```

Input parameters:

<i>AxisId</i> (INT)	Axis identifier [1...64]
<i>Mode</i> (INT)	Type of selection

Execution mode:

Wait

Use:

Range selection

Each spindle axis is associated with 4 gear ranges (configured in ODM). Function AX_SelectGearFoller makes it possible to select the desired range. The 4 ranges available can be selected by means of the *Mode* parameter. The range set with this function remains active until the next range change command or until system shutdown. When the system is restarted, range 1 is activated.

Servo error

Servo error is the parameter, associated with the servo mode, which defines the maximum deviation admissible between the Theoretical point and the actual position of the axis; this value depends on type of movement. There are three types of servo error, selectable with the *Mode* parameter: movement with VFF, movement without VFF and StandBy. The execution of this function entails the activation of the relative servo error values configured in ODM. The types of servo error (and the relative values) set with this function remain active until the axis is moved (by a process or by WinPLUS); before starting the movement, the system will change the servo error by reactivating the mode configured in ODM.

Mode	Value	Meaning
SET_GEAR1	1	Range 1 selection
SET_GEAR2	2	Range 2 selection
SET_GEAR3	3	Range 3 selection
SET_GEAR4	4	Range 4 selection
SET_MOV_WITHVFF	5	Selection of servo with VFF for moving axis
SET_MOV_NOVFF	6	Selection of servo without VFF for moving axis
SET_STANDBY	7	Selection of STAND_BY servo error for stationary axis

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors.
0x00200001	Not enough memory for processing.
0x00200003	Too many requests to axis board.
0x00200004	AxisId wrong.
0x00200006	AxisId not configured.
0x00200060	Type of servo error wrong or Mode parameter not envisaged.

Example:

```
Ret : dword;  
Ret := AX_SelectGearFoller (1, 2);
```

Selects range 2 for the axis with ID 1.

7.28 AX_SpindleSwitch

Function AX_SpindleSwitch allows to change the mode of spindle use activating axis characterization.

Syntax:

```
ret_code := AX_SpindleSwitch (AxisId, Mode) ;
```

Input parameters:

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>Mode</i> (DWORD)	Mode to be activated [0..1]

Execution mode:

Wait

Use:

This function is used to change the mode of spindle use. Mode = 1 makes it possible to update the axis parameterization configured in ODM, Mode = 0 returns the spindle parameterization. In this case Range 1 is selected automatically with the speed controlled spindle.

Mode =	0x00000000	SPINDLE_Mode	Sets spindle mode
	0x00000001	AXIS_Mode	Sets axis mode

The function returns error if spindle/axis is moving. Before using axis, it is advisable to home the axis.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200003	Too many requests to axes board
0x00200004	AxisId wrong
0x00200004	AxisId wrong.
0x00200006	Axis not configured
0x0020000F	Call to virtual axis or not spindle
0x00200062	Moving axis
0x00200084	Parameter Mode out of range

Example:

```
Ret : dword;
Ret := AX_SpindleSwitch (1, 1);
```

7.29 AX_CalibDisable

AX_CalibDisable function disables all offsets generated by an axis.

Syntax

```
ret_code := AX_CalibDisable (AxisId) ;
```

Input parameters

AxisId (INT) Axis id [1..64]

Execution mode

Wait

Use

This function disables the calculation and the application of all offsets generated by the *AxisId* axis. Following the function execution also the axis offset will be disabled.

Return values

The following table lists all possible values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200006	Axis not configured
0x00200016	Instruction already active
0x00200062	Axis in motion
0x00200017	No conditions to enable the command

Example

```
Ret : dword;  
Ret := AX_CalibDisable (1);
```

Offset is disabled on axis with ID=1.

See also

[AX_CalibDisableGet](#), [AX_CalibEnable](#), [AX_CalibEnableGet](#), [AX_CalibLoadFile](#), [AX_CalibStatus](#)

7.30 AX_CalibEnable

The AX_CalibEnable function enables an axis in receiving all offsets available.

Syntax

```
ret_code := AX_CalibEnable (AxisId) ;
```

Input parameter

AxisId (INT) Axis id[1..64]

Execution mode

Wait

Use

Following the execution of the function, any axis having offset contributions referred to the axis specified, can apply this contribution.

Return values

The following table lists all possible values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200006	Axis not configured

Example

```
Ret : dword;  
Ret := AX_CalibEnable (1);
```

All the offsets applied on the axis with ID=1 are disabled.

See also

[AX_CalibDisableGet](#), [AX_CalibDisable](#), [AX_CalibEnableGet](#), [AX_CalibLoadFile](#), [AX_CalibStatus](#)

7.31 AX_CalibLoadFile

The AX_CalibLoadFile function uploads the offset file for an axis.

Syntax

```
ret_code := AX_CalibLoadFile (AxisId, FilePath) ;
```

Input parameters

AxisId (INT)	Axis identifier [1..64]
FilePath (STRING)	PathName of the file containing offsets data

Execution mode

Wait

Use

This function uploads the offsets file related to an axis.

The execution of this function does not change the active offsets status. If the application of the offset data contribution from this axis towards others was active, it won't change its status. If contributions given by this axis were disabled, they keep on being disabled. The *AX_CalibEnable* function enables the offset contributions.

Return values

The following table lists all possible values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200006	Axis not configured
0x00200001	Available memory expired
0x0020000A	File not found
0x00200097	Error in the first line of the offsets file
0x00200098	Error in the second line of the offsets file
0x00200099	Error in a line containing data in the offsets file
0x0020009A	Error in analysing the offsets file

Example

```
Ret : dword;
Ret := AX_CalibLoadFile (1,'\\ssd\\osai\\xtend\\oem\\comptab\\tab1') ;
```

The offset table specified is uploaded for axis with ID=1.

See also

[AX_CalibDisable](#), [AX_CalibDisableGet](#), [AX_CalibEnable](#), [AX_CalibEnableGet](#), [AX_CalibStatus](#)

7.32 AX_CalibDisableGet

The AX_CalibDisableGet function disables for an axis, the offsets made by another axis.

Syntax

```
ret_code := AX_CalibDisableGet (AxisId, AxisIdGet) ;
```

Input parameters

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>AxisIdGet</i> (INT)	Axis identifier generating offset [1..64]

Execution mode

Wait

Use

The function disables the offset contributions receipt for one axis (cross-offset, squaring errors offset) due to one or other axes.

If the two call parameters are equal, the axis offset itself will be disabled (pitch-screw offset, rotations offsets, squaring errors offset).

Return values

The following table lists all possible values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200006	Axis not configured
0x00200016	Command already active
0x00200062	Axis in motion
0x00200017	No conditions to enable the command

Example

```
Ret : dword;
Ret := AX_CalibDisableGet (1, 3);
```

On the axis with ID=1 the offset due to the motion with ID=3 axis is disabled.

See also

[AX_CalibDisable](#), [AX_CalibEnable](#), [AX_CalibEnableGet](#), [AX_CalibLoadFile](#), [AX_CalibStatus](#)

7.33 AX_CalibEnableGet

For one axis, the function AX_CalibEnableGet enables the offsets made by another axis.

Syntax

```
ret_code := AX_CalibEnableGet (AxisId, AxisIdGet) ;
```

Input parameters

AxisId (INT) Axis identifier [1..64]

AxisIdGet (INT) Axis identifier generating the offset [1..64]

Execution mode

Wait

Use

For a given axis this function enables the receipt of the compensation contributions for another axis.

If programming two parameters having the same value, the offset of the axis itself is enabled.

Return values

The following table lists all possible values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200006	Axis not configured

Example

```
Ret : dword;  
Ret := AX_CalibEnableGet (1, 3);
```

On the axis with ID=1 the offset due to the motion with ID=3 axis is disabled.

See also

[AX_CalibDisableGet](#), [AX_CalibDisable](#), [AX_CalibEnable](#), [AX_CalibLoadFile](#), [AX_CalibStatus](#)

7.34 AX_CalibStatus

The AX_CalibStatus function returns the Word status active offsets on an axis.

Syntax

```
ret_code := AX_CalibStatus (AxisId, Status) ;
```

Input parameter

AxisId (INT) Axis identifier [1..64]

Output parameter

Status (WORD) StatusWord contains the active offsets status

Execution mode

Wait

Use

This function allows to know which type of offsets contributions can be generated by the specified axis.

The status word contains offset types generated by the specified axis and encoded as shown in the The following table:

Status	Description
0x0001	Axis offset itself (self-offset)
0x0002	Cross-compensation
0x0004	Squaring errors offset

Return value

The following table lists all possible *ret_code* variable can have:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200006	Axis not configured

Example

```
Ret : dword;
Sts : word ;
Ret := AX_CalibStatus (1, Sts);
```

For axis with ID=1 its offset status is returned in Sts.

See also

[AX_CalibDisable](#), [AX_CalibDisableGet](#), [AX_CalibEnable](#), [AX_CalibEnableGet](#), [AX_CalibLoadFile](#)

7.35 AX_ServoToPLC

The AX_ServoToPLC function allows the servo to communicate with the PLC.

Syntax

```
ret_code := AX_ServoToPLC (AxisId, AxReq) ;
```

Input parameter

AxisId (INT) Axis identifier [1..64]

Output parameter

AxReq (AxReq_Struct) Servo requests to the PLC

Execution mode

Immediate

Use

This function allows the servo to communicate with the PLC.

The AxisId parameter defines the axis involved.

The AxReq parameter, having an AxReq_Struct structure type, contains the servo requests to the PLC and the PLC feedback to the servo. AxReq_Structstructure fields are servo written and read by the PLC.

This is the AxReq_Struct structure definition:

Field	Type	Description
ReqDW	DWORD	Servo requests to the PLC
AnsDW	DWORD	Servo feedback to the PLC requests
Data	ARRAY[0..5] OF LREAL	Data array, not used so far

The ReqDW field is a bit mask having the following meaning:

Bit	Description
0	When bit passes from 0 to 1, servo asks the PLC if the probing managing is available
1	When bit passes from 0 to 1, servo asks the PLC to enable the probing When bit passes from 1 to 0, servo asks the PLC to disable/abort/stop the probing
2	If bit is high, servo asks the PLC the position where the probing occurred
3	If bit is high, servo asks the PLC if probe is pressed
4	If bit is high, servo asks the PLC if the probing occurred
5..15	Not used (set to 0)

The AnsDW field is a bit mask having the following meaning:

Bit	Description
0	Servo sets at 1 this bit after aborting the probing following a PLC abort request
1..15	Not used (set to 0)

Return values

The following table lists all possible that *ret_code* variable can have:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	Wrong AxisId.
0x00200006	Axis not set

Example

```
Ret : dword;
AxReq: AxReq_Struct;

Ret := AX_ServoToPLC (1, AxReq);
```

The PLC reads the servo request concerning the axis with ID=1.

See also

[AX_PLCToServo](#)

7.36 AX_PLCToServo

The AX_PLCToServo function allows the PLC to reply the servo requests.

Syntax

```
ret_code := AX_PLCToServo (AxisId, AxAns) ;
```

Input parameters

<i>AxisId</i> (INT)	Axis identifier [1..64]
<i>AxAns</i> (AxAns_Struct)	PLC feedback to servo requests

Execution mode

Immediate

Use

This function allows the PLC to reply to the servo requests.

The AxisId parameter defines the axis involved. The AxAns parameter, having an AxAns_Struct structure type, contains the PLC feedback to the requests of the servo and to the PLC requests towards the servo. All fields of the AxAns_Struct structure are written by the PLC and read by the servo.

This is the definition of the AXAns_Struct:

Field	Type	Description
AnsDW	DWORD	PLC feedback to servo requests
ReqDW	DWORD	PLC feedback towards the servo
Data	ARRAY[0..5] OF LREAL	The first element of the Data array contains the probing position, important only after the probing process. All other array elements are not used so far..

The AnsDW field is a bit mask having the following meaning:

Bit	Description
0	High if PLC can manage the probing
1	High if probing is enabled
2	Not used (set to 0)
3	High if probe is pressed, low when probe is not pressed
4	High when “probing done” event occurs
5..15	Not used (set to 0)

The ReqDW field is a bit mask having the following meaning:

Bit	Description
0	The PLC sets at 1 this bit when asks the servo to abort probing
1..15	Not used (set at 0)

Return values

The following table lists all values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	<i>AxisId</i> wrong.
0x00200006	Axis not set

Example

```

Ret1, Ret2 : dword;
AxReq: AxReq_Struct;
AxAns: AxAns_Struct;
AxisId : int := 1;

Ret1 := AX_ServoToPLC(AxisId, AxReq);
if and(AxReq.ReqDW, 16#1) > 0 then
    AxAns.AnsDW := AxAns.AnsDW or 16#1;
end_if;
Ret2 := AX_PLCToServo(AxisId, AxAns);

```

The PLC reads the servo requests referred to the axis with ID=1.

When servo asks if the PLC is available to manage the probing, PLC feedback is that it is available.

7.37 RS_CreateResource

The RS_CreateResource function allows to configure a new D/A, A/D or transducer resource.

Syntax:

```
ret_code := RS_CreateResource (ResourceType, CfgStruct, Resourceld) ;
```

Input parameters:

<i>ResourceType</i> (WORD)	Type of resource [1..3]
<i>CfgStruct</i> (ResCfgType)	Parameters structure

Output parameter:

<i>Resourceld</i> (INT)	Logic identifier or resource
-------------------------	------------------------------

Execution mode:

Wait.

Use:

This function configures a resource not used yet by the axes board. The resource could be a D/A or A/D converter, an encoder transducer, a sensor or an axis. Encoder transducers and D/A converters of the devices on Bridge OS-Wire and EtherCAT Bridge A664/A432 field buses can be also configured.

The type of resource to be configured is specified in parameter *ResourceType* with the following meaning:

ResourceType	Value	Type of resource
AXISRD_TYPE_TRANSDUCER	1	Transducer
AXISRD_TYPE_ANALOG_OUT	2	D/A converter
AXISRD_TYPE_ANALOG_INP	3	A/D converter
AXISRD_TYPE_SENSOR	4	Sensor
AXISRD_TYPE_AXIS	6	Axis

Resource identification is by means of the parameters in structure *CfgStruct* :

Field	Description
<i>Board</i>	Indicates the primary(0) and secondary (1) boards
<i>Interface</i>	Type of interface
<i>Node</i>	Bus node (for analog resources, a value of 0 corresponds to the local board, otherwise it denotes a remote node on the field bus)
<i>Modulus</i>	Modulus number in case of modular devices with bus coupler
<i>Index</i>	Indicates the position within the node or the modulus
<i>Parameter1</i>	See following specific cases in the manual
<i>Parameter2</i>	In the case of transducer configuration see below, otherwise it is reserved.
<i>Parameter3</i>	In the case of transducer configuration see below, otherwise it is reserved.
<i>Parameter4</i>	Reserved.

For possible values to be assigned to *Interface*, *Node*, *Modulus* and *Index* files, see “WinPLUS Library - User manual” paragraph 5.42 “Resources mapping within the Servo loop ambient”

In case of transducer configuration (only on local boards and OS-Wire Bridge) parameters have the following values:

<i>Parameter1</i>	Electrical pitch
<i>Parameter2</i>	Configuration of transducer channel with code: <ul style="list-style-type: none"> ➢ Bit 0: A channel polarity reversal (**) ➢ Bit 1: B channel polarity reversal(**) ➢ Bit 2: Z channel polarity reversal(**) ➢ Bit 3: count direction reversal ➢ Bit 4: marker signal detect on level(**) ➢ Bit 5: marker signal detect on front(**) ➢ Bit 6: reserved ➢ Bit 7: broken wire detect enable (*) ➢ Bits from 8 to 14: reserved ➢ Bit 15: broken wire detect activation (*)
<i>Parameter3</i>	Refresh time in microseconds (max 65536, it has to be a multiplier of the servo basic clock)

(*) To activate the broken wire detect function, set both bits 7 and 15 to 1.

(**) Not available for EtherCAT Bridge A664/A432

In case of D/A convertor configuration (only on local boards or OS-Wire Bridge) the first parameter has the following meaning:

<i>Parameter1</i>	Refresh time in microseconds (max 65536, it has to be a multiplier of the servo basic clock)
-------------------	--

In case of A/D convertor configuration (only on local boards OS2005) the first parameter has the following meaning:

<i>Parameter1</i>	Refresh time in microseconds (max 65536, it has to be a multiplier of the servo basic clock)
-------------------	--

In case of sensor setting (only on OS2011 and OS2020 boards) first parameter has the following value:

<i>Parameter1</i>	Refresh time in microseconds (max 65536, it has to be a multiplier of the servo basic clock)
-------------------	--

In case of axis configuration, parameters can have the following meanings:

<i>Parameter1</i>	Axis type. Only virtual axis type is accepted
<i>Parameter2</i>	Servo clock in ms
<i>Parameter3</i>	Interpolation clock

If the call has a favourable outcome, a logic ID identifier that enables the resource to be used through calls to specified functions is returned in variable Resourceld.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200001	Not enough memory to execute the operation.
0x00200003	Too many requests to the axis board.
0x00200011	Resources already allocated.
0x00200012	Max. number of configurable resources reached.
0x00200080	One of the parameters is wrong.
0x00250001	EtherCAT not found in ENI file

Example for OS2005 board:

```

Ret : dword;
ResId : word;
CfgSt : ResCfgType;
CfgSt.Board := 0; (* primary board*)
CfgSt.Interface := ANALOG_INTERFACE; (* analogic *)
CfgSt.Node := 0; (* local on board *)
CfgSt.Module := 0; (* N/A *)
CfgSt.Index := 9; (* first D/A at 16 bits *)
CfgSt.Parameter1 := 2000; (* refresh at 2 milliseconds *)

Ret := RS_CreateResource (AXISRD_TYPE_ANALOG_OUT, CfgSt, ResId);

```

Configures the first converted D/A at 16 bits of the board.

See also

[RS_DestroyResource](#)

7.38 RS_DestroyResource

The RS_DestroyResource function deletes a D/A, A/D resources or a transducer previously set with the function RS_CreateResource.

Syntax:

```
ret_code := RS_DestroyResource (Resourceld, ResourceType) ;
```

Input parameters:

Resourceld (INT) Logical ID of the resource

ResourceType (WORD) source type

Execution mode

Wait

Description

This function releases a D/A or A/D of transducer previously configured with the RS_CreateResource function. For each resource type only one deletion at a time is allowed. It is not possible to delete transducers and converters associated to an axis.

The resource type to be deleted is indicated in the *ResourceType* parameter having the following meaning:

ResourceType	Value	ResourceType
AXISRD_TYPE_TRANSDUCER	1	Transducer
AXISRD_TYPE_ANALOG_OUT	2	D/A converter
AXISRD_TYPE_ANALOG_INP	3	A/D converter
AXISRD_TYPE_SENSOR	4	Sensor
AXISRD_TYPE_AXIS	6	Axis

Return code

ret_code can have the following values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200003	Too many requests to the axis board
0x00200004	Resourceld parameter out of the validity range
0x0020000F	Transducer/converter resource is associated to an axis
0x00200011	Resource already allocated
0x00200012	ResourceType parameter out of the validity range
0x00200016	A deletion is already active, try again
0x0020005F	Operation impossible to execute (timeout)

Example

```
Ret : dword;
ret_code := RS_DestroyResource (1, AXISRD_TYPE_ANALOG_OUT);
```

Deletes the D/A converter having an ID =1.

See also

RS_CreateResource.

7.39 RS_SensorExecCmd

The RS_SensorExecCmd function executes commands on the sensor.

Syntax

```
ret_code := RS_SensorExecCmd (SensId, Cmd, Value, Info) ;
```

Input parameters

<i>SensId</i> (INT)	Sensor Identifier [1..64]
<i>Cmd</i> (INT)	Command number [1..100]
<i>Value</i> (INT)	Command value

Output parameters

<i>Info</i> (ResSensInfo_Struct)	Real time information
----------------------------------	-----------------------

Execution mode

Wait

Description

This function executes following instructions

Id	Mnemonic	Value
1	SN_CMD_MODE When the distance measurement is disabled, it is possible to force the sensor feedback with the RS_SensorWritePar function.	0 - frequency measurement disabled 1 - frequency measurement enabled
2	SN_CMD_ENTAB 1<= N <= NUM_SENS_TAB If the table N does not exist, the number of active table will not change. N = 0 -> Disables the active table N > NUM_SENS_TAB -> the function returns error	N - Table to enable
3	SN_CMD_DELTAB	N - Table to delete
4	SN_CMD_RESET	Resets the sensor
5	SN_CMD_INFO It copies in the output structure Info the real-time information.	-

WORD	SN_CMD_HWEMG = 16#0011 (Hardware tip touch emergency) SN_CMD_SWQEMG = 16#0012 (Software quick tip protection) SN_CMD_SWNEMG = 16#0014 (Software normal tip protection) SN_CMD_HOLEEMG = 16#0018 (Avoid hole protection) SN_CMD_TIPEMG = 16#0020 (Tip damages protection) It is possible to enable/disable both emergencies	0 - Emergency management enabled 1 - Emergency management disabled
100	SN_CMD_FREQ	0 - Frequency integrated circuit measurement disabled 1 - Frequency integrated circuit measurement enabled 2 - Internal capacitance selection 3 - External capacitance selection

(*)

ResSensInfo_Struct Field	Description
dist	Distance measurement (mm)
freq	Frequency measurement (Hz)
temp	Oscillator temperature
plasmaAC	Plasma AC signal (V)
plasmaDC	Plasma DC signal (V)
registry	Registry configured with the RS_SensorWritePar function
status	Sensor status (**)
acttab	Active table number

(**)

Sensor Status	Description
0x00000001	Hardware tip touch condition
0x00000002	Sensor Idle
0x00000004	Sensor Ready (frequency measurement enabled)
0x00000008	Sensor Initialized (there is an active table)
0x00000010	Emergency status

Return codePossible *ret_code* values:

ret_code (HEX)	Description
0x00000000	Function executed without error
0x00200004	Invalid SensId.
0x0020001b	Sensor not configured.
0x00200080	Invalid Cmd.
0x00200018	Timeout error

Example

```
Ret : dword;  
Ret := RS_SensorExecCmd (1, cmd_SENSOR_MODE, 1);
```

Enables the frequency measurement on sensor with ID 1.

See also

[RS_SensorReadPar](#), [RS_SensorWritePar](#), [RS_SensorReadTab](#), [RS_SensorWriteTab](#)

7.40 RS_SensorReadPar

The RS_SensorReadPar function reads sensor parameters.

Syntax

```
ret_code := RS_SensorReadPar (SensId, Param, Value) ;
```

Input parameters

SensId (INT) Sensor Identifier [1..64]
Param (INT) Parameter number

Output parameters

Value (LREAL) Parameter value

Execution mode

Wait.

Description

This function must be used to read following parameters:

Param	Mnemonic	Measurement units
1	sn_STANDOFF_TRG(*)	M.U. / mm
2	sn_STANDOFF_FBK	M.U. / mm
3	sn_SSZ_RADIUS(*)	M.U. / mm
4	sn_SSZ_CENTREX(*)	M.U. / mm
5	sn_SSZ_CENTREY(*)	M.U. / mm
6	sn_SSZ_CENTREZ(*)	M.U. / mm
7	sn_AXIS_ID1(*)	
8	sn_AXIS_ID2(*)	
9	sn_AXIS_ID3(*)	
10	sn_OFFSET_CALIB	M.U. / mm
11	sn_REG_TO_MONIT	
12	sn_TIMER_HW_EMG	M.U. / us
13	sn_TIMER_SWQ_EMG	M.U. / us
14	sn_TIMER_SWN_EMG	M.U. / us
15	sn_SENS_THRESHOLD	M.U. / mm
16	sn_TIMER_TIP_EMG	M.U. / us
17	sn_TIP_THRESHOLD	M.U. / MHz
200+n	sn_REGISTRY	

M.U. means “Measuring Unit”.

The symbol (*) indicates that the parameter is stored in the sensor area but it is only used by the PLC application.

If the sensor is disabled, it is possible to force its feedback with the sn_STANDOFF_FBK parameter.

sn_STANDOFF_TRG: it is the standoff target.

sn_REGISTRY: it allows the user to read/write Hermes08 board registries; n is the registry number (see the Hermes08 documentation)

sn_SSZ_RADIUS, sn_SSZ_CENTREX, sn_SSZ_CENTREY, sn_SSZ_CENTREZ: they are configuration Shadow Zone parameters.

sn_AXIS_ID1, sn_AXIS_ID2 e sn_AXIS_ID3: they are the identifiers of axes associated with the sensor.

sn_OFFSET_CALIB: it is the offset to apply to the sensor calibration curve when external conditions(i.e. temperature) change.

sn_TIMER_HW_EMG, sn_TIMER_SWQ_EMG, sn_TIMER_SWN_EMG e sn_TIMER_TIP_EMG: they are the timers for the sensor protections management. Each protection has its owner timer. sn_TIMER_HW_EMG is used for the hardware tip touch emergency (It should be disabled during the calibration procedure).

sn_TIMER_SWQ_EMG and sn_TIMER_SWN_EMG are used for the software protection: the sensor feedback is compared with the configured threshold (sn_SENSOR_THRESHOLD) trying to prevent machine damages.

sn_TIMER_MAX_FREQ is used to detect tip damages. When the sensor feedback exceeds the configured threshold(sn_TIP_THRESHOLD) the emergency occurs.

sn_MEASURE_FREQ: returns the measurement of the frequency integrated circuit.

Return values

Possible *ret_code* values

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200004	Invalid SensId
0x00200028	Sensor not configured
0x00200080	Invalid Param.

Example

```
Ret : dword;
PARM_VALUE : lreal;
Ret := RS_SensorReadPar (1, sn_READSSZRADIUS, PARM_VALUE);
```

Reads the shadow zone radius configured for the sensor with ID 1.

See also

RS_SensorWritePar, RS_SensorWriteTab, RS_SensorReadTab, RS_SensorExecCmd

7.41 RS_SensorWritePar

The RS_SensorWritePar function writes sensor parameters.

Syntax

ret_code := RS_SensorWritePar (*SensId*, *Param*, *Value*) ;

Input parameters

<i>SensId</i> (INT)	Sensor Identifier [1..64]
<i>Param</i> (INT)	Parameter number
<i>Value</i> (LREAL)	Parameter value

Execution mode

Wait.

Description

This function must be used to write the following parameters:

Param	Mnemonic Name	Measurement units
1	sn_STANDOFF_TRG(*)	M.U. / mm
2	sn_STANDOFF_FBK	M.U. / mm
3	sn_SSZ_RADIUS(*)	M.U. / mm
4	sn_SSZ_CENTRE1(*)	M.U. / mm
5	sn_SSZ_CENTRE2(*)	M.U. / mm
6	sn_SSZ_CENTRE3(*)	M.U. / mm
7	sn_AXIS_ID1(*)	
8	sn_AXIS_ID2(*)	
9	sn_AXIS_ID3(*)	
10	sn_OFFSETS_CALIB	M.U. / MHz
11	sn_REG_TO_MONIT	
12	sn_TIMER_HW_EMG	M.U. / ms
13	sn_TIMER_SWQ_EMG	M.U. / ms
14	sn_TIMER_SWN_EMG	M.U. / ms
15	sn_SENS_THRESHOLD	M.U. / mm
16	sn_TIMER_TIP_EMG	M.U. / ms
17	sn_TIP_THRESHOLD	M.U. / MHz
200+n	sn_REGISTRY	

M.U. means “Measuring Unit”.

The symbol (*) indicates that the parameter is stored in the sensor area but it is only used by the PLC application.

If the sensor is disabled, it is possible to force its feedback with the sn_STANDOFF_FBK parameter.
sn_STANDOFF_TRG: it is the standoff target.

sn_REGISTRY: it allows the user to read/write Hermes08 board registries; n is the registry number (see the Hermes08 documentation).

`sn_SSZ_RADIUS`, `sn_SSZ_CENTREX`, `sn_SSZ_CENTREY`, `sn_SSZ_CENTREZ`: they are used to configure the sensor Shadow Zone.

`sn_AXIS_ID1`, `sn_AXIS_ID2` e `sn_AXIS_ID3`: they are the identifiers of axes associated with the sensor.

`sn_OFFSET_CALIB`: it is the offset to apply to the sensor calibration curve when external conditions(i.e. temperature) change.

`sn_TIMER_HW_EMG`, `sn_TIMER_SWQ_EMG`, `sn_TIMER_SWN_EMG` and `sn_TIMER_TIP_EMG`: they are the timers for the sensor protections management. Each protection has its owner timer. `sn_TIMER_HW_EMG` is used for the hardware tip touch emergency (It should be disabled during the calibration procedure).

`sn_TIMER_SWQ_EMG` and `sn_TIMER_SWN_EMG` are used for the software protection: the sensor feedback is compared with the configured threshold (`sn_SENSOR_THRESHOLD`) trying to prevent machine damages.

`sn_TIMER_TIP_EMG` is used to detect tip damages . When the sensor feedback exceeds the configured threshold(`sn_TIP_THRESHOLD`) the emergency occurs.

Return values

Possible `ret_code` values.

<code>ret_code (HEX)</code>	Description
0x00000000	Function executed without error
0x00200004	Invalid SensId
0x00200028	Sensor not configured
0x00200080	Invalid Param
0x00200006	Axis not configured

Example

```
Ret : dword;
PARM_VALUE : lreal;
Ret := RS_SensorWritePar (1, sn_READSSZRADIUS, PARM_VALUE);
```

Writes the shadow zone radius on the sensor with ID 1.

See also

`RS_SensorReadPar`, `RS_SensorWriteTab`, `RS_SensorReadTab`, `RS_SensorExecCmd`

7.42 RS_SensorReadTable

The RS_SensorReadTable function reads the sensor calibration table.

Syntax

```
ret_code := RS_SensorReadTable (SensId, Param, Tab) ;
```

Input parameters

<i>SensId</i> (INT)	Sensor identifier [1..64]
<i>Param</i> (INT)	Table number [1..10]

Output parameters

<i>Value</i> (ResSensTable_Struct)	Calibration table
------------------------------------	-------------------

Execution Mode

Wait

Description

This function uses the *ResSensTable_Struct* structure (see the following description) to read the calibration table.

Field	Description
InterpMeth	Interpolation method 0 = linear interpolation 1 = cubic spline interpolation
NPnt	Number of fields filled (<=30)
Temperature	Temperature data acquisition (°C)
X[30]	Table data (i.e. frequency)
Y[30]	Table data (i.e. distance)

Return values

Possible *ret_code* values.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without error
0x00200004	Invalid SensId
0x00200028	Sensor not configured.
0x00200080	Invalid Param.
0x0020002a	Table not created

Example

```
Ret : dword;
Tab: ResSensTable_Struct;
Ret := RS_SensorReadTable (1, 1 , Tab);
```

Reads the table 1 of the sensor with ID 1.

See also:

[RS_SensorWritePar](#), [RS_SensorWriteTable](#), [RS_SensorReadPar](#), [RS_SensorExecCmd](#)

7.43 RS_SensorWriteTable

The RS_SensorWriteTable function writes the sensor calibration table.

Syntax

```
ret_code := RS_SensorWriteTable (SensId, Param, Value) ;
```

Input parameters

<i>SensId</i> (INT)	Sensor identifier [1..64]
<i>Param</i> (INT)	Table number [1..10]
<i>Value</i> (ResSensTable_Struct)	Calibration table

Execution mode

Wait.

Description

This function stores the calibration table in the sensor area by means of the *ResSensTable_Struct* structure (see the following description) and calculates the coefficients of the interpolating function. The temperature information is useful to check when you can apply the calibration curve. The sensor can store up to 10 tables but you can enable only a table at a time (with the *RS_SensorExeCommand* function).

Field	Description
InterpMeth	Interpolation method 0 = linear interpolation 1 = cubic spline interpolation
NPnt	Number of fields filled (<=30)
Temperature	Temperature data acquisition (°C)
X[30]	Table data (i.e. frequency)
Y[30]	Table data (i.e. distance)

Return values

Possible *ret_code* values.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without error
0x00200004	Invalid SensId
0x00200028	Sensor not configured.
0x00200080	Invalid Param
0x0020002b	Table already created
0x003a0003	Table not strictly monotonic

Example

```
Ret : dword;
Tab : ResSensTable_Struct;
Ret := RS_SensorWriteTable (1, 1, Tab);
```

Fills the table 1 for the sensor with ID 1.

See also

[RS_SensorReadPar](#), [RS_SensorWritePar](#), [RS_SensorReadTable](#), [RS_SensorExecCmd](#)

7.44 RS_TransducerRead

Function RS_TransducerRead reads the count of a local or remote encoder transducer on an OS-Wire Bridge.

Syntax:

```
ret_code := RS_TransducerRead (TrasId, Value) ;
```

Input parameters:

<i>TrasId</i> (INT)	Transducer identifier [1..64]
---------------------	-------------------------------

Output parameters:

<i>Value</i> (LREAL)	Read value
----------------------	------------

Execution mode:

Immediate

Use:

This function reads the physical position of the specified transducer. If the transducer hasn't been associated to an axis or to a hand wheel, it is necessary to use the RS_CreateResource function before reading.

The output position is expressed in number of impulses of the transducer.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	TrasId wrong.
0x00200013	Transducer not configured.

Example:

```
Ret : dword;
Tpos : lreal;
Ret := RS_TransducerRead (1, Tpos);
```

Reads the position of the transducer with ID=1.

See also:

[RS_CreateResource](#).

7.45 RS_AnalogRead

Function RS_AnalogRead reads a local D/A converter, on the axis board.

Syntax:

```
ret_code := RS_AnalogRead (Convid, Value) ;
```

Input parameters:

<i>Convid</i> (INT)	A/D converter identifier [1..32]
---------------------	----------------------------------

Output parameters:

<i>Value</i> (LREAL)	Read value
----------------------	------------

Execution mode:

Immediate

Use:

This function reads the value of the local A/D converter on axes board. If the converter has been calibrated as voltage input, V is the unit of measure, if it has been calibrated as current input, the Ampere is the unit of measure.

Actual resolution levels depend on type of converter:

for 12-bit A/D converters, minimum input increment is 4.883 mV.

To obtain a *Convid* identifier, use function RS_CreateResource.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00200004	Convid wrong.
0x00200014	Converter not configured.

Example:

```
Ret : dword;
Val : lreal;
Ret := RS_AnalogRead (1, Val);
```

Reads the analog input with ID=1.

See also:

[RS_CreateResource](#).

7.46 RS_AnalogWrite

Function RS_AnalogWrite writes values on a local D/A converter, on an axis board or an OS-Wire Bridge.

Syntax:

```
ret_code := RS_AnalogWrite (ConvId, Value) ;
```

Input parameters:

ConvId (INT)	D/A converter identifier [1..32]
Value (LREAL)	Value to be written [-10..+10]

Execution mode:

NoWait

Use:

Function RS_AnalogWrite writes values on a local D/A converter, on an axis board or an OS-Wire Bridge. The converter must not be assigned to an axis. V is the unit of measure; if the values written are out of range, they are automatically cut to the limit values.

Actual resolution levels depend on type of converter:

- For 14 bit D/A an increment in the output corresponds to 1.22 mV.
- For 8 bit D/A an increment in the output corresponds to 78,125 mV.

To obtain a ConvId identifier, use function RS_CreateResource.



This function does not perform any specific checks on the analog output of the converter. Make sure that the values are written in a correct manner and in accordance with the devices to be managed.

The analog output of a converter is updated during the execution time of a servo tick.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00200004	ConvId wrong.
0x00200014	Converter not configured.

Example:

```
Ret : dword;
Ret := RS_AnalogWrite (1, -5.4);
Write -5.4 Volt on the converter with ID=1.
```

See also:

[RS_CreateResource](#).

7.47 List of errors returned by Servo environment

Servo errors table

Error code (HEX)	Description
0x00200001	Not enough memory to execute the operation
0x00200002	MBOX channels exhausted
0x00200003	Too many requests to axis board
0x00200004	Resource Id out of range
0x00200005	Wrong DLL upload
0x00200006	Axis Id not present
0x00200007	Axis board not found
0x00200008	Error in accessing axis data configuration file
0x00200009	Error in accessing extended data configuration file
0x0020000A	File not present
0x0020000D	Request made on non-digital axis
0x0020000E	Request made on non analog axis
0x0020000F	Invalid request (generic, depends on function)
0x00200010	Invalid resource type
0x00200011	Resource already allocated
0x00200012	Resource not available
0x00200013	Transducer Id not present
0x00200014	Converter Id not present
0x00200015	Bootstrap stopped due to emergency
0x00200016	Command already active
0x00200017	Conditions insufficient to execute command
0x00200018	Command execution timeout
0x00200020	Monitoring already active
0x00200021	Monitoring not active
0x00200022	Monitoring function non-existent
0x00200023	Undefined variable
0x00200024	Wrong variables number
0x00200028	Missing sensor Id
0x00200029	Id sensor table wrong
0x00200030	Master slave definition error
0x00200031	Operation not possible on slave
0x00200032	Function valid only for gantry axes
0x00200033	Gantry axis in skew error
0x00200034	Conversion of offset between markers value wrong (for gantry axes)
0x00200035	Function not valid for gantry axes
from 0x00200040 to 0x0020005F	Internal errors
0x00200060	Servo error type selection wrong
0x00200061	Servo entity queue full
0x00200062	Interpolation active
0x00200063	Servo type selection wrong

0x00200064	Probe anomaly
0x00200065	Filter not configured
0x00200066	Filter Id invalid
0x00200067	Invalid command on filter
0x00200070	Timeout during communication with digital drive
0x00200071	Communication with digital drive channel busy
0x00200072	Axis inhibited due to severe error on digital drive
0x00200073	Stage of communication with digital drive not valid
0x00200074	Node not found on BUS
0x00200075	Communication channel inhibited due to severe error on digital drive
0x00200076	Invalid axis use
0x00200077	Invalid timeout value
0x00200080	Invalid parameter (generic)
0x00200081	Ids of axes not congruent
0x00200082	KCKV conversion wrong (overflow on long)
0x00200083	Error in calculation of tolerance space
0x00200084	Value out of range (generic)
0x00200085	Electrical pitch too large
0x00200086	Transducer not configured
0x00200087	Gear number wrong
0x00200088	Digital axis not available for exchange
0x00200089	Axis shared on exclusive basis
0x0020008A	Axis board wrong
0x0020008B	Overflow in calculation of max. speed
0x0020008C	Invalid parameter type
0x0020008D	Converter not set
0x00200093	Error in opening ALT file
0x00200094	Error in opening APL file
0x00200095	Syntax error in APL or ALT file
0x00200096	ALT file not complete with active axes
0x00200097	Error 1 in the offset files analysis
0x00200098	Error 2 in the offset files analysis
0x00200099	Error in offsets measure columns
0x0020009A	Error 3 in the offset files analysis
0x0020009B	Drive parameters not checked
0x002000A0	Axis in servo error
0x002000A1	Axis in tolerance timeout error
0x002000A2	Axis in tolerance space error
0x002000A3	Marker too far from micro
0x002000A4	Broken wire detected in encoder channel
0x002000A5	Encoder broken wire

Table of servo errors sub-classification OS-Wire*Internal errors*

Error code (HEX)	Description
0x00210001	Overflow in remote node data access buffer
0x00210002	OS-Wire hardware not recognised
0x00210003	Drive or Node not found on bus
0x00210004	Error in accessing firmware file (OS3 drive)
0x00210005	Error during firmware recording (OS3 drive)
0x00210006	Conversion error for factor = 0 (OS3 drive)
0x00210007	Too many read/write parameters (OS3 drive)
0x00210008	Max_Torque = 0 (OS3 drive)
0x00210009	AMotMax = 0 drive (OS3)
0x0021000A	Error in determination of OS3 drive speed loop parameters
0x0021000B	Error in determination of tuning parameters
0x0021000C	Drive size not congruent
0x0021000D	Motor part number not congruent
0x0021000E	Parameter congruence test cannot be executed
0x0021000F	Invalid request (generic)
0x002100A0	OS3 drive alarm

OS3 drive errors

Error code (HEX)	Description
0x00210101	E01 Command cannot be executed
0x00210102	E02 Overflow in monitoring table
0x00210103	E03 Monitoring already active
0x00210104	E04 No variable to be monitored
0x00210105	E05 Error during transmission
0x00210106	E06 Timeout error in transmission
0x00210107	E07 Timeout error in reception
0x00210108	E08 Checksum error
0x00210109	E09 Echo error
0x0021010A	E10 Hiperface error
0x0021010B	E11 Overflow in feedback table
0x0021010C	E12 Variable Id wrong
0x0021010D	E13 Overflow in parameters table
0x0021010E	E14 Parameter Id wrong
0x0021010F	E15 Parameter write not congruent
0x00210110	E16 Memory address not valid
0x00210111	E17 Incongruent data in Sequencer
0x00210112	E18 Command parameters not valid
0x00210113	E19 Point2Point command not available
0x00210114	E20 Offset identification not available
0x00210115	E21 Synchronisation timeout on RTC2
0x00210116	E22 FPGA register Id wrong

Error code (HEX)	Description
0x00210117	E23 Overflow in number of locations
0x00210118	E24 Qep counter release timeout
0x00210119	E25 External Qep Td counter release timeout
0x0021011A	E26 No encoder interface connected
0x0021011B	E27 Power error

OS3 drive alarms

Error code (HEX)	Description
0x0021021E	A30 Braking resistance circuit error
0x0021021F	A31 Watchdog DSP
0x00210220	A32 NMI triggered
0x00210222	A34 Initial position wrong
0x00210228	A40 Drive over temperature
0x00210229	A41 Motor over temperature or thermal protection
0x0021022A	A42 Overcurrent or short-circuit
0x0021022B	A43 Braking resistance overcurrent
0x0021022C	A44 Braking resistance disconnected
0x0021022D	A45 Over/undervoltage protection
0x0021022F	A47 Bus overvoltage
0x00210230	A48 Bus undervoltage
0x00210231	A49 Intervention of I2t motor
0x00210232	A50 Intervention of I2t braking resistance
0x00210233	A51 Intervention of I2t drive
0x00210234	A52 Rapid Halt request
0x00210235	A53 Encoder faulty or not connected
0x00210236	A54 Emergency stop timeout
0x00210237	A55 Following error out of tolerance
0x00210238	A56 Speed error out of tolerance
0x0021023A	A58 Alarm from power board
0x0021023B	A59 Parameter wrong
0x0021023C	A60 Power outage
0x0021023D	A61 SinCos error
0x0021023E	A62 Calibration over travel
0x0021023F	A63 Igbt brake closing timeout
0x00210240	A64 External Safety Device active
0x00210241	A65 Current request cannot be implemented
0x00210242	A66 Motor position cannot be controlled
0x00210243	A67 Motor speed cannot be controlled
0x00210244	A68 Collision
0x00210245	A69 External position error

Servo errors SERCOS III and SoE sub-classes table*Internal errors*

Error code (HEX)	Description
0x00220001	Drive not found on the bus
0x00220002	Invalid data to access IDn parameter
0x00220003	Invalid IDn element
0x00220004	Data not provided by Service Channel
0x00220005	Service Channel data are cut (alert)
0x00220007	Error when enabling the drive
0x00220008	Invalid IDN type
0x00220020	IDN S-0-0146 command error
0x00220021	IDN P-0-0014 command error
0x00220022	IDN S-0-0172 command error
0x00220023	IDN S-0-0170 command error
0x00220024	IDN S-0-0405 command error
0x002200A0	SERCOS drive alarm

Servo errors table (CANopen subclass)*Internal errors*

Error code (HEX)	Description
0x00230002	Drive to be rebooted
0x00230003	Drive not found on the bus
0x00230004	Error in writing parameters
0x00230005	Drive not set
0x00230006	No feedback from the drive
0x002300A0	CANopen drive alarm

Servo errors table (Sensor subclass)*Internal errors*

Error code (HEX)	Description
0x00240001	Contact between the sensors and the part
0x00240002	“Quick Protection” threshold exceeded
0x00240003	“Normal Protection” threshold exceeded
0x00240004	Sensor not working properly
0x00240005	“Hole” detected

Servo errors table (EtherCAT subclass)*Internal errors*

Error code (HEX)	Description
0x00250001	Node not found on the bus

Servo errors table (Mechatrolink subclass)*Internal errors*

Error code (HEX)	Description
0x00260001	Node not found on the bus
0x00260002	SENS_ON command error
0x00260003	SYNC_SET command error
0x00260004	SMON command error
0x00260005	WDOG error
0x00260006	Generic command error
0x00260007	Maximum dwell time expired for general command execution
0x00260008	Warning when reading/writing the drive parameter
0x00260009	Parameters difference detected between CNC and drive
0x0026000A	Drive alarm when rebooting
0x0026000B	Communication alarm
0x002600A0	Mechatrolink drive alarm

7.48 Resources mapping within the Servo loop ambient

Introduction

This chapter describes the definition of D/A, A/D converters and Encoder transducers on the boards and on the PE devices used by the servo loop mode task. From now on, these objects are defined as “resources”.

Generalities

Resources are identified by a logical ID that is an integer valid with the class category. The ID of a resource is configured in the ODM or it can be generated by the system through a resource loading function. When CNC is working resources can't be allocated to release the ID.

Within the system, every resource has an univocal identification through its “logical address” containing the following information:

All resources are identified by a logic ID, given by an integer and valid within the relational set. Every resource must always be used through the ID, set in the ODM configuration or given by the allocation resources function RS_CreateResource.

Each resource has an univocal identification through its “Logical Address” made by following information:

- a) *Board*. Board number (0=Primary, 1=Secondary)
- b) *Interface*. Type of fieldbus interface using the following codes:

Interface	Value	Interface type
ANALOG_INTERF	0	Analogic interface
SERCOS_INTERF	1	Sercos interface
MECHAT_INTERF	2	Mechatrolink interface
OSWIRE_INTERF	3	OSWire interface
CANOPE_INTERF	4	CANOpen interface
CANA10_INTERF	5	Analog CANopen A10 axis
CANP24_INTERF	6	Analog CANopen P24 axis
ETNHMS_INTERF	7	ETN Hermes08 interface
ETCSOE_INTERF	8	EtherCAT SoE
ETCNLG_INTERF	9	EtherCAT Analog A664/A432
ETCCOE_INTERF	10	EtherCAT CoE
ETCPED_INTERF	11	EtherCAT Pulse&Direction P664/P432

- c) *Node*. Node address, 0=local board, from 1 to 61 on filed bus
- d) *SubModule*. Modulus number in case of modular devices with bus coupler, 0=single modulus (or compact)
- e) *Index*. Position, from 1 to 65535, in the node, i.e. the resource position within the device

Information has to be given when allocating dynamically a resource, otherwise is set when the resource is configured with the ODM. Resources are divided mainly as:

- ▷ Local resources (allocated on Axes boards)
- ▷ Remote resources (allocated on remote devices of field buses)

In case of analogic resources, Node=0 indicates that the resource is local on the board and node 0 can't be used to address remote nodes.

Local resources

OPEN systems local resources are grouped in resources “logical blocks” of the same type. According to the CNC models, each box has a number of resources, starting from zero up to the maximum number defined by the each box. Therefore, for the three classes of resources we have:

- ▷ A/D converters are grouped in two boxes with 4 resources each, from 1 to 4 in the first box and from 5 to 8 in the second.
- ▷ D/A converters are divided in two boxes with 8 resources each, from 1 to 8 in the first and from 9 to 16 in the second.
- ▷ Encoder transducers, as there is only one type, they are mapped in a single box, from 1 to 8.

Index (resources numeration) within the boxes represents the method used to characterize the resources.

Resources available

Local resources, OS8365 board

ResourceType	Resources number	Index
AXISRD_TYPE_TRANSDUCER	1	1
AXISRD_TYPE_ANALOG_OUT	2	1,2 (first and second D/A 16 bit)
AXISRD_TYPE_ANALOG_INP	2	1,2 (first and second A/D)

Local resources, OS2005 board

ResourceType	Resources number	Index
AXISRD_TYPE_TRANSDUCER	1	1
AXISRD_TYPE_ANALOG_OUT	1	1 (first D/A 16 bit)
AXISRD_TYPE_ANALOG_INP	2	1,2 (first and second A/D)

Remote resources

For remote resources mapping (Node > 0) see following tables:

OS-Wire OS3 drive(Interface = OSWIRE_INTERF):

ResourceType	Resources number	Index
AXISRD_TYPE_TRANSDUCER	2	1,2 (motor transducer, external transducer)
AXISRD_TYPE_ANALOG_OUT	0	-
AXISRD_TYPE_ANALOG_INP	0	-

OS-Wire Bridge device (*Interface* = ANALOG_INTERF):

ResourceType	Resources number	Index
AXISRD_TYPE_TRANSDUCER	3	1,2,3
AXISRD_TYPE_ANALOG_OUT	3	1,2,3
AXISRD_TYPE_ANALOG_INP	0	-

EtherCAT Analog Bridge A664 (*Interface* = ETCNLG_INTERF) device:

ResourceType	Resources number	Index
AXISRD_TYPE_TRANSDUCER	7	Da 1 a 7
AXISRD_TYPE_ANALOG_OUT	7	Da 1 a 7
AXISRD_TYPE_ANALOG_INP	4	N/A (not servo managed)

EtherCAT Analog Bridge A432 (*Interface* = ETCNLG_INTERF) device:

ResourceType	Resources number	Index
AXISRD_TYPE_TRANSDUCER	5	Da 1 a 5
AXISRD_TYPE_ANALOG_OUT	7	Da 1 a 7
AXISRD_TYPE_ANALOG_INP	4	N/A (not servo managed)

EtherCAT P&D Bridge P664 (*Interface* = ETCPED_INTERF) device:

ResourceType	Resources number	Index
AXISRD_TYPE_TRANSDUCER	7	Da 1 a 7
AXISRD_TYPE_ANALOG_OUT	1	7
AXISRD_TYPE_ANALOG_INP	4	N/A (not servo managed)

EtherCAT P&D Bridge P432 (*Interface* = ETCPED_INTERF) device:

ResourceType	Resources number	Index
AXISRD_TYPE_TRANSDUCER	5	From 1 to 5
AXISRD_TYPE_ANALOG_OUT	1	7
AXISRD_TYPE_ANALOG_INP	4	N/A (not servo managed)

8. 4C_CANopen LIBRARY: CANopen FUNCTIONS

CANopen_BoardCmd	Stops and restarts data transmission via the CANopen network
CANopen_GetEmcyInfo	Reads emergency conditions returned by a node that is part of the configuration
CANopen_GetEmcyNode	Makes it possible to test or return a node containing diagnostic data
CANopen_GetStatus	Returns some data regarding the status of the nodes included in the CANopen network
CANopen_NmtCmd	Sends nmt commands via the CANopen network
CANopen_ReadSdoCmd	Reads an object in the object dictionary of a node
CANopen_SyncCmd	Transmits a sync signal via the CANopen network
CANopen_WriteSdoCmd	Writes an object in the object dictionary of a node

8.1 CANopen_BoardCmd

The CANopen_BoardCmd function is used to stop and restart data transmission via CANopen network.

Syntax:

```
ret_code := CANopen_BoardCmd (ExecMode, ExecStatus, Board, Command, CfgFileName) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_WAIT]
<i>Board</i> (INT)	Master board where the function operates. For PRIMAlogic this parameter must be 0
<i>Command</i> (INT)	Specifies type of command to be sent
<i>CfgFileName</i> (STRING)	Specifies file where the CANopen network configuration is described. If the string contained in this parameter is blank, the default configuration file is used. This parameter is significant only if the value of the Command parameter is cmd_START

Output parameters:

<i>ExecStatus</i> (DWORD)	Execution status of command in MODE_NOWAIT_ACK mode
---------------------------	---

Execution mode:

Set with *ExecMode*

Use:

This function executes operations that modify communication status as a function of the *Command* parameter, which may take on the following values:

Command	Values	Description
cmd_RESET	1	CANopen communication RESET
cmd_START	2	CANopen communication START
cmd_STOP	3	CANopen communication STOP

The **RESET** command drops all CANopen communications and hence the status of remote nodes may change to WatchDog status. To reactivate the communications at the end of the RESET command the START command must always be given.

The **START** command restores CANopen communications and executes the complete “Node Bootup” sequence defined in the configuration file.

The **STOP** command stops all CANopen activities on the BUS. **PRIMAlogic does not support this command.**

In STOP status and during a RESET operation, the status of remote nodes may change to WatchDog status.

Return values:

Value (HEX)	Description
0	Function executed successfully
0x00350004	Invalid command
0x00350020	DRIVER Error during internal initialisation (driver error)
0x00350032	DEVICE RESET command execution timeout (driver error)
0x00350033	DEVICE COM-flag not set (driver error)
0x0035003E	USER IOCTL function failed (driver error)
0x0035003F	Invalid board user parameter

Example:

```
ExecStatus : dword;
```

```
Ret : dword;
```

```
Ret := CANopen_BoardCmd(MODE_WAIT, ExecStatus, 0, cmd_RESET, ''');
```

```
(* drops CANopen communications. *)
```

8.2 CANopen_GetEmcyInfo

The CANopen_GetEmcyInfo function reads emergency conditions returned by a node that is part of the configuration.

Syntax:

```
ret_code := CANopen_GetEmcyInfo (ExecMode, ExecStatus, Board, NodeID, Timeout, NodeStatus,
                                 AddlInfo, ProfileNum, NodeState, ActualErr, EmcyLen,
                                 EmcyData);
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>Board (INT)</i>	Master board where the function operates. For PRIMALogic it must be 0
<i>NodeID (INT)</i>	Identifier of the node whose diagnostic data is to be read
<i>Timeout (INT)</i>	Timeout in ms (value 0, means no timeout)

Output parameters:

<i>ExecStatus (DWORD)</i>	Execution status of command in mode MODE_NOWAIT_ACK
<i>NodeStatus (WORD)</i>	Node status (bit template)
<i>AddlInfo (WORD)</i>	Two bytes read by node during start-up. Extensive info on type of node (input, output, etc.)
<i>ProfileNum (INT)</i>	Two bytes read by node during start-up. Number of CANopen device profile
<i>NodeState (INT)</i>	Node state if guarding protocol is active on node
<i>ActualErr (INT)</i>	Current error in node. Returned during boot phase.
<i>EmcyLen (INT)</i>	Number of emergencies in emergency array
<i>EmcyData (ARRAY of STRUCT)</i>	Array of emergency structures containing diagnostic data. Each structure contains the data returned by the CANopen standard emergency message

Execution mode:

MODE_WAIT: This function remains waiting for response from a remote node, unless a timeout period has been specified; in this case, if no response is received, it returns a timeout error.

Use:

This function returns some data on the diagnostics of the node specified in parameter *NodeID*.

The *NodeStatus* output parameter is comprised of several bits; the meaning of the individual bits is as follows:

NodeStatus	Bit	Description
st_NO_RESPONSE	0x0001	Node does not respond
st_EMCY_BUFF_OVFL	0x0002	Emergency buffer overflow
st_PRM_FAULT	0x0004	Difference between master and node configurations
st_GUARD_ACTIVE	0x0008	Node guarding protocol active on this node
st_DEACT	0x0080	Node is deactivated and it is not managed by master

The *ProfileNum* output parameter contains the number of the CANopen device profile associated with the node. This parameter may have different values, the most common are the following:

ProfileNum	Values	Description
dp_IO_MODULUSS	401	Device Profile for I/O moduluss
dp_DRIVES	402	Device Profile for Drives and Motion Control
dp_ENCODERS	406	Device Profile for Encoders

The *NodeState* output parameter, which contains the NMT status of the node (if the guarding protocol is active on the node), may take on the following values:

NodeState	Values	Description
ns_DISCONNECTED	1	Disconnected
ns_CONNECTING	2	Connecting
ns_PREPARING	3	Preparing
ns_PREPARED	4	Prepared
ns_OPERATIONAL	5	Operational
ns_PREOPERATIONAL	127	Pre-operational

The *ActualErr* output parameter may take on the following values:

Values	Description
30	Guarding failed
31	Node has changed status and is no longer operational
32	Sequence error in guarding protocol
33	No response to remote request from configured PDO
34	No response from node during its configuration
35	Node profile number defined in master configuration not the same as node profile number
36	Node device type defined in master configuration not the same as node device type
37	Response from unknown SDO received
38	Length of message received by an SDO is not 8
39	Node not managed, node in stop

The CANopen_EmcyType structure contains the data associated with CANopen emergency messages; this structure is comprised of the following fields:

Field	Type	Description
ErrorCode	WORD	Error code of emergency; associated with CANopen standard 1003h object
ErrorRegister	BYTE	Error register; associated with CANopen standard 1001h object
ManufacturerError	DWORD	Possible error code notified by node manufacturer

Return values:

Value (HEX)	Description
0	Function executed successfully
0x00350006	Node number not compatible
0x0035003F	Invalid Board parameter

Example:

```
VAR
  ExecMode : INT := MODE_WAIT;
  ExecStatus : DWORD;
  NodeStatus : WORD;
  AddInfo : WORD;
  ProfileNum : INT;
  NodeState : INT;
  ActualErr : INT;
  EmcyLen : INT;
  EmcyData : ARRAY [0..4] of CANopen_EmcyType;

END_VAR
Ret_code := CANopen_GetEmcyInfo(ExecMode, ExecStatus, Board, NodeId, 0,
                                NodeStatus,     AddInfo,     ProfileNum,     NodeState,
                                ActualErr,    EmcyLen,    EmcyData);
```

8.3 CANopen_GetEmcyNode

The CANopen_GetEmcyNode function makes it possible to test or return a node containing diagnostic data (see CANopen_GetEmcyInfo function).

Syntax:

```
ret_code := CANopen_GetEmcyNode(ExecMode, ExecStatus, Board, NodeID);
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>Board (INT)</i>	Master board where the function operates. For PRIMAlogic it must be 0
<i>NodeID (INT)</i>	Identifier of node to be tested

Output parameters:

<i>ecStatus (DWORD)</i>	Execution status of command in mode MODE_NOWAIT_ACK
<i>NodeID (INT)</i>	Identifier of node returned by function

Execution mode:

MODE_WAIT: This function remains waiting for a response from a remote node, unless a timeout period has been specified; in this case, if no response is received, it returns a timeout error.

Use:

If parameter *NodeID* is zero and there is at least one node in error, the function returns the first node in error in *NodeID*.

If the value of the *NodeID* parameter is other than zero, instead, the function determines whether or not this node is in error.

Return values:

Value (HEX)	Description
0	No node in error
0x00350005	Node is in error
0x0035003F	Invalid <i>board</i> parameter

Example:

```
VAR
  ExecStatus : DWORD;
  Board : INT;
  NodeID : INT;
  ret_code : DWORD ;

  ret_code := CANopen_GetEmcyNode(MODE_WAIT, ExecStatus, Board, NodeID);
```

8.4 CANopen_GetStatus

The CANopen_GetStatus function returns some data regarding the status of the nodes included in the CANopen network.

Syntax:

```
ret_code := CANopen_GetStatus(ExecMode, ExecStatus, Board, StatusData);
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [only MODE_WAIT]
<i>Board</i> (INT)	Master board where the function operates. For PRIMAlogic it must be 0

Output parameters

<i>ExecStatus</i> (DWORD)	Execution status of command in mode MODE_NOWAIT_ACK.
<i>StatusData</i> (STRUCT)	This CANopen_StatusType structure contains the status of the nodes in the configuration

Execution mode:

This function can be executed only in MODE_WAIT mode.

Use:

The function reads some information regarding the status of the CANopen master board and about which nodes are configured, active or possibly in error in the CANopen network.

All this information is grouped in the CANopen_StatusType structure, which is comprised of the following fields:

Field	Type	Description
<i>NetStatus</i>	WORD	The value of this field is not significant.
<i>MasterStatus</i>	WORD	Status of CANopen master board. See table with possible values of this field further on.
<i>ConfigNode</i>	ARRAY [0..15] of BYTE	Identifies the slave nodes present in the CANopen master board configuration. Each node is associated with a bit.
<i>ActiveNode</i>	ARRAY [0..15] of BYTE	Identifies slave nodes that are currently active. Each node is associated with a bit.
<i>DiagnNode</i>	ARRAY [0..15] of BYTE	Identifies slave nodes currently in error. Each node is associated with a bit.

The *MasterStatus* field of the CANopen_StatusType structure may have the following values:

MasterStatus	Values (HEX)
ms_OFFLINE	0x0000
ms_STOP	0x0040
ms_CLEAR	0x0080
ms_OPERATE	0x00C0

The *ConfigNode* field (16-byte array), which is part of the CANopen_StatusType structure, contains the information as to which slave nodes are present in the CANopen master board configuration: each node is associated with a bit in the byte array. The following table shows the association between bits and Nodelds:

ConfigNode	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	7	6	5	4	3	2	1	0
Byte 1	15	14	13	12	11	10	9	8
Byte 2	23	22	21	20	19	18	17	16
...								
Byte 15	127	126	125	124	123	122	121	120

For instance, the node with address 6 is associated with bit 6 of byte 0 of the *ConfigNode* array.

If the bit is 1, then the corresponding node is present in the master configuration; if the bit is 0, then the corresponding node is not present in the master configuration.

The *ActivNode* field (16-byte array), which is part of the CANopen_StatusType structure, contains the information as to which slave nodes are active: each node is associated with a bit in the byte array. The following table shows the association between bits and Nodelds:

ActivNode	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	7	6	5	4	3	2	1	0
Byte 1	15	14	13	12	11	10	9	8
Byte 2	23	22	21	20	19	18	17	16
...								
Byte 15	127	126	125	124	123	122	121	120

For instance, the node with address 23 is associated with bit 7 of byte 2 of the *ActivNode* array.

If the bit is 1, then the corresponding node is active, node guarding detects no error; if the bit is 0, then the corresponding node is not active, either because it is not configured or because it is in error.

The *DiagnNode* field (16-byte array), which is part of the CANopen_StatusType structure, contains a diagnostic bit for each node. The following table shows the association between bits and Nodelds:

DiagnNode	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	7	6	5	4	3	2	1	0
Byte 1	15	14	13	12	11	10	9	8
Byte 2	23	22	21	20	19	18	17	16
...								
Byte 15	127	126	125	124	123	122	121	120

For instance, the node with address 120 is associated with bit 0 of byte 15 of the *DiagnNode* array.

If the bit is 1, it means that a new emergency message has been received and has not yet been read by the CANopen_GetEmcyInfo function; if the bit is 0, then no new emergency message has been received.

Return values:

Value (HEX)	Description
0	Function executed successfully
0x0035003F	Invalid <i>Board</i> user parameter

Example:

VAR

```
RetCode : DWORD ;
ExecMode : INT := MODE_WAIT;
ExecStatus : DWORD;
Board : INT;
StatusData : CANopen_StatusType;

END_VAR

RetCode := CANopen_GetStatus(ExecMode, ExecStatus, Board, StatusData);
```

8.5 CANopen_NmtCmd

The CANopen_NmtCmd function sends NMT commands via the CANopen network.

Syntax:

```
ret_code := CANopen_NmtCmd (ExecMode, ExecStatus, Board, NodId, Cmd, Timeout) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [only MODE_WAIT]
<i>Board</i> (INT)	Master board where the function operates. For PRIMAlogic it must be 0
<i>NodId</i> (INT)	Node identifier (a value of 0 means broadcasting)
<i>Cmd</i> (INT)	NMT command
<i>Timeout</i> (INT)	Timeout in ms (a value of 0 means no timeout). Not significant for PRIMAlogic

Output parameters

<i>ExecStatus</i> (DWORD)	Execution status of command in mode MODE_NOWAIT_ACK
---------------------------	---

Execution mode:

This function can be executed only in MODE_WAIT mode.

Use:

This function is used to execute network management services (NMT: Network Management). Through these services, the nodes can be initialised, sent, stopped and reset.

Parameter *NodId* (1-127) identifies the node where to execute an NMT service. If you want to execute the service on all the nodes present on the BUS, set *NodId* to 0.

Parameter *Cmd* indicates the type of service to be executed. Possible values are:

Cmd	Values	Description
nmt_START_REMOTE_NODE	1	Start remote node
nmt_STOP_REMOTE_NODE	2	Stop remote node
nmt_ENTER_PREOPERATIONAL	128	Set pre-operational status
nmt_RESET_NODE	129	Reset remote node
nmt_RESET_COMMUNICATION	130	Reset communications

Return**values:**

Value (HEX)	Description
0	Function executed successfully
0x00350001	Message transmission timeout (only in WAIT mode)
0x00350002	Message reception timeout (only in WAIT mode)
0x00350004	NMT command wrong
0x00350005	Command rejected (internal commands list full)
0x00350006	Node not compatible

Example:

In this example, the NMT service “Start remote node” is executed on node with address 10.

```
RetDword : dword;
ExecStatus : dword;
Board : int;
NodeId : int;
RetDword     :=     CANopen_NmtCmd(MODE_WAIT,      ExecStatus,      Board,      NodeId,
nmt_START_REMOTE_NODE, 0);
```

8.6 CANopen_ReadSdoCmd

The CANopen_ReadSdoCmd function reads an object in the Object Dictionary of a node.

Syntax:

```
ret_code := CANopen_ReadSdoCmd(ExecMode, ExecStatus, Board, NodeID, Index, subIndex,  
                  Timeout, DataLength, Data);
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>Board</i> (INT)	Master board where the function operates. Not significant for PRIMAlogic
<i>NodeID</i> (INT)	Node identifier
<i>Index</i> (WORD)	Index of object dictionary object to be read
<i>subIndex</i> (INT)	SubIndex of object dictionary object to be read
<i>Timeout</i> (INT)	Timeout in ms (0 = no timeout). Not significant for PRIMAlogic

Output parameters

<i>ExecStatus</i> (DWORD)	Execution status of command in mode MODE_NOWAIT_ACK
<i>DataLength</i> (INT)	Length of data received
<i>Data</i> (ANY_ELEMENTARY)	Data received

Execution mode:

MODE_WAIT:

This function remains waiting for response from a remote node, unless a timeout period has been specified; in the latter case, if no response is received, it returns a timeout error. If timeout value is =0, an error could occur as the system does not have time to get a feedback on the command sending outcome.

MODE_NOWAIT_ACK:

unction does not remain waiting for response from a remote node. The first time it makes the request, the other times it makes sure a reply has been received, otherwise it remains waiting or quits with a timeout error.

For PRIMAlogic this function can only be executed in MODE_NOWAIT_ACK mode, while OPENcontrol supports only other modes.

Use:

This function makes it possible to read the information contained in an object in the Object Dictionary of a node. The object is specified in parameters **Index** and **subIndex**.

In the **Index** parameter you must write the hexadecimal value of the index of the object, whereas its **subIndex** must be specified in the **subIndex** parameter.

In the **DataLength** output parameter the function returns the length of the data contained in the object and in the **Data** parameter it returns the object data.

Return values:

Value (HEX)	Description
0	Function executed successfully
0x00350001	Message transmission timeout (Only in WAIT mode)
0x00350002	Message reception timeout (Only in WAIT mode)
0x00350003	Service has been rejected by node with Abort SDO. The cause could be invalid Index and subIndex parameters, or user not in possession of node access rights.
0x00350004	Command wrong
0x00350005	Commands list full (WAIT mode)
0x00350006	Master board CAN chip in OFFLINE status
0x00350007	Limits of reception buffer exceeded.
0x00350008	Fragmented protocol data exceed reception buffer limits.
0x00350009	Previous service still active and not confirmed to application.
0x0035000A	Sequence error in fragmenting protocol. Request aborted.
0x0035000B	No response from node or node not present.
0x0035000C	Node is not in operational status and access is denied. At present SDO channel is being used by master board to configure the node during start stage. Try to request service again.
0x00350031	No message available



PRIMAlogic only supports expedited SDOs (up to 4 data bytes).

Example:

In this example, object with index 16#2010, subIndex 2 of node 6 is read.

```

VAR
    Datalen : int;
    Data : int;
    RetCode : dword;

ReadExecStatus : word;
    ReadCompletedWithSuccess : bool := false;
    ReadCompletedWithError : bool := false;
    SdoUnderWay : bool := true;
;
END_VAR

IF SdoUnderWay THEN
    RetCode := CANopen_ReadSdoCmd(MODE_NOWAIT_ACK, ReadExecStatus, 0, 6,
                                16#2010, 2, 0, Datalen, Data);
    IF ReadExecStatus = 16#80000000 THEN
        (*SDO Read in progress *)
        ReadCompletedWithSuccess := false;
        ReadCompletedWithError := false;
    ELSIF ReadExecStatus = 0 THEN

```

```
(* Successful SDO read; now Data variable contains data read through SDO *)
    ReadCompletedWithSuccess := true;
    SdoUnderWay := false;
ELSE
    ReadCompletedWithError := true; (* SDO read completed with error *)
    SdoUnderWay := false;
END_IF;
END_IF;
```

8.7 CANopen_SyncCmd

The CANopen_SyncCmd function transmits a SYNC signal via the CANopen network.

Syntax:

```
ret_code := CANopen_SyncCmd(ExecMode, ExecStatus, Board, Timeout);
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_WAIT]
<i>Board</i> (INT)	Master board where the function operates. Not significant for PRIMAlogic
<i>Timeout</i> (INT)	Timeout in ms (0 = no timeout). Not significant for PRIMAlogic

Output parameters:

<i>ExecStatus</i> (DWORD)	Execution status of command in mode MODE_NOWAIT_ACK
---------------------------	---

Execution mode:

This function can be executed only in MODE_WAIT mode.

Use:

Call this function when you want to send a SYNC signal via the CANopen network.

This command can be used to update the data to/ from analog modules.

Return values:

Value (HEX)	Description
0	Function executed successfully
0X00350004	Command rejected
0x00350005	Commands list full (in WAIT mode)

Example:

```
Retdword : dword;
ExecStatus : dword;
Retdword := CANopen_SyncCmd(MODE_WAIT, ExecStatus, 0, 0);
```

8.8 CANopen_WriteSdoCmd

The CANopen_WriteSdoCmd function allows to write an object in the Object Dictionary of a node.

Syntax:

```
ret_code := CANopen_WriteSdoCmd (ExecMode, ExecStatus, Board, NodId, Index, subIndex,
Timeout, DataLength, Data);
```

Input parameters:

<i>ExecMode (INT)</i>	Command execution mode [MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>Board (INT)</i>	Only MODE_NOWAIT_ACK mode is supported by PRIMAlogic
<i>NodId (INT)</i>	Master board where the function operates. Not significant for PRIMAlogic
<i>Index (WORD)</i>	Node identifier
<i>subIndex (INT)</i>	Index of object dictionary object to be written
<i>Timeout (INT)</i>	SubIndex of object dictionary object to be written
<i>DataLength (INT)</i>	Timeout in ms (0 = no timeout). Not significant for PRIMAlogic
<i>Data (ANY_ELEMENTARY)</i>	Data length to be written in object
<i>Data (ANY_ELEMENTARY)</i>	Data to be written in object

Output parameters

<i>ExecStatus (DWORD)</i>	Execution status of command in mode MODE_NOWAIT_ACK
---------------------------	---

Execution mode:

MODE_WAIT:

This function remains waiting for a response from a remote node, unless a timeout period has been specified; in the latter case, if no response is received, it returns a timeout error. If timeout value is =0, an error could occur as the system does not have time to get a feedback on the command sending outcome.

MODE_NOWAIT_ACK:

Function does not remain waiting for a response from remote node. The first time it makes the request, the other times it makes sure a reply has been received, otherwise it remains waiting or quits with a timeout error.

For PRIMAlogic this function can be executed only in MODE_NOWAIT_ACK mode, while OPENcontrol only supports other mode.

Use:

This function makes it possible to write information in an Object Dictionary object of a node. The object is specified in parameters **Index** and **subIndex**. In the **Index** parameter you must write the hexadecimal value of the index of the object, whereas its subIndex must be specified in the **subIndex** parameter.

Return values:

Value (HEX)	Description
0	Function executed successfully
0x00350001	Message transmission timeout (Only in WAIT mode)
0x00350002	Message reception timeout (Only in WAIT mode)
0x00350003	Service has been rejected by node with Abort SDO. The cause could be invalid Index and subIndex parameters, or user not in possession of node access rights.

Value (HEX)	Description
	<i>Function output data contain the Abort SDO code.</i>
0x00350004	Command wrong
0x00350005	Commands list full (in WAIT mode)
0x00350006	Master board CAN chip in OFFLINE status
0x00350007	Limits of reception buffer exceeded.
0x00350008	Fragmented protocol data exceed reception buffer limits.
0x00350009	Previous service still active and not confirmed to application.
0x0035000A	Sequence error in fragmenting protocol. Request aborted.
0x0035000B	No response from node or node not present.
0x0035000C	Node is not in operational status and access is denied. At present SDO channel is being used by master board to configure the node during start stage. Try to request service again.
0x00350031	No message available



PRIMAl*ogic* only supports expedited type SDOs (up to 4 data bytes).

Example:

In this example, object with index 16#2010, subIndex 1 of node 6 is written.

VAR

```

    RetCode : dword;
    WriteExecStatus : dword;
    WriteData : byte;
    WriteDataLen : int;
    WriteCompletedWithSuccess : bool := false;
    WriteCompletedWithError : bool := false;
    SdoUnderWay : bool := true;
END_VAR

```

```

IF SdoUnderWay THEN
    WriteData := 6; (* this is the value that will be written *)
    WriteDataLen := 1; (* index object 16#2010, bibindex 1 is 1 byte long *)
    RetCode := CANopen_WriteSdoCmd(MODE_NOWAIT_ACK, WriteExecStatus, 0, 6,
                                    16#2010, 1, 0, WriteDataLen, WriteData);

```

```

IF WriteExecStatus = 16#80000000 THEN
    (*SDO write in progress *)
    WriteCompletedWithSuccess := false;
    WriteCompletedWithError := false;
ELSIF WriteExecStatus = 0 THEN
    (*SDO write completed successfully *)
    WriteCompletedWithSuccess := true;
    SdoUnderWay := false;
ELSE
    (*SDO write completed with error *)

```

```
    WriteCompletedWithError := true;  
    SdoUnderWay := false;  
  END_IF;  
END_IF;
```

9. Functions for the management of XML files - 4C_XMLFILE

XMLClose	Closes an xml file
XMLOpen	Opens an xml file
XMLRead	Reads one or more data from the elements of an xml file
XMLWrite	Writes one or more data in the elements of an xml file
XMLReadAttribute	Reads one or more data from the attributes of an xml file
XMLReadAttributeEx	Reads one or more data from the attributes of an XML file
XMLReadEx	Reads one or more data of the XML file elements
XMLWriteAttribute	Writes one or more data in the attributes of an xml file
XMLRemove	Removes an element from an xml file
XMLRemoveAttribute	Removes an attribute from an xml file
XMLSave	Saves an xml file in file system

9.1 XMLClose

The XMLClose function closes an XML file.

Syntax:

```
ret_code := XMLClose (XMLhandle) ;
```

Input parameters:

XMLhandle (DINT) Handle to an XML file already opened

Execution mode:

Wait

Use:

This function closes an XML file and free the related memory space.

The XML file to be closed is identified by parameter *XMLhandle*: this parameter is returned by the XMLOpen function upon the opening of the file.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XMLhandle parameter

Example:

```
Ret : dword;
XMLhandle : dint;
IntVar : int;
(* opens file SSD\osai\file.xml in read only mode, reads the attribute of the
top element in the IntVar variable, and finally closes the file with function
XMLClose *)
Ret := XMLOpen ('\SSD\osai\file.xml', XML_FMODE_READ, XMLhandle) ;
Ret := XMLReadAttribute( XMLhandle, '/top', 'attrib', 'IntVar');
Ret := XMLClose(XMLhandle) ;
```

9.2 XMLOpen

The XMLOpen function opens an XML file.

Syntax:

```
ret_code := XMLOpen (Path, Mode, XMLhandle) ;
```

Input parameters:

<i>Path</i> (STRING)	Full pathname of the file to be opened
<i>Mode</i> (INT)	File opening mode

Output parameter:

<i>XMLhandle</i> (DINT)	Univocal identifier of the XML file opened
-------------------------	--

Execution mode:

Wait

Use:

This function lets you open an XML file and store it in the memory. This function returns output parameter *XMLhandle*, which is a unique identifier of the XML file opened; the *XMLhandle* parameter must be used whenever a file is to be accessed in read or write mode.

Mode parameter is a bits mask with the following meaning:

Bit	Description and possible values
0..1	0,0 : reserved 0,1 : XML_FMODE_READ (file is opened in read only mode and uploaded in the memory) 1,0 : XML_FMODE_READ_WRITE (file is opened in read/write mode and uploaded in the memory) 1,1: XML_FMODE_CREATE (File is created from zero and it is a read only file II file)
2..3	Not used (set =0)
4	XML_FMODE_ERR_IF_ENCODING_UNKN: this bit is important if file is opened in read only mode or in read/write mode. If set =1: if XML file encoding cannot be converted, then it displays error If set =0: if XML file encoding cannot be converted, then file is read without any conversion
5..15	Not used (set =0)

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000001	Invalid Path parameter
0x00000004	Invalid Mode parameter
0x00000005	Memory allocation error
0x00000006	Path parameter too long
0x00000007	Error in loading file
0x0000000D	Too many XML files opened

Example:

```
Ret : dword;
XMLhandle : dint ;
...
(* opens file \SSD\osai\file.xml in read only mode and saves it in the memory *)
Ret := XMLOpen ('\\SSD\\osai\\file.xml', XML_FMODE_READ, XMLhandle) ;
```

9.3 XMLRead

The XMLRead function reads one or more data from the elements of an XML file

Syntax:

```
ret_code := XMLRead (XMLhandle, XmlPath, VarName) ;
```

Input parameters:

<i>XMLhandle</i> (DINT)	univocal XML file identifier
<i>XmlPath</i> (STRING)	XML path of elements from which data have to be read
<i>VarName</i> (STRING)	PLC variable where to write the data read in XML file

Output parameter:

Execution mode:

Wait

Use:

This function reads data from the elements of an XML file and write them in a variable, called *VarName*, defined in the PLC program. The XML file from which the data are to be read is identified by parameter *XMLhandle*, this parameter is returned by the XMLOpen function upon the opening of the file. Parameter *XmlPath* lets you select an element in the XML file from which the data are to be read; if the element identified by the *XmlPath* parameter does not exist in the XML file, then the function returns an error. Depending on the type of *VarName* variable, this function lets you read a single scalar value or an array of scalar values, or a structure that may even be nested. A structure may even contain arrays of structures, provided that the following restriction is complied with: the lowest value of the array index must be 1, e.g., StructArray: ARRAY[1..10] OF Mystruct;

The function searches for the *VarName* function from among the symbols and determines what type it is.

The *VarName* variable type can be one of the following:

- ANY_ELEMENTARY or ARRAY of ANY_ELEMENTARY type. In this case the function reads the data from the XML file and writes the data directly to variable *VarName*. The type of *VarName* variable must be compatible with the type of data associated with the element identified by parameter *XmlPath*.
- STRUCT. The function creates a list of the ANY_ELEMENTARY type or ARRAY of ANY_ELEMENTARY type variables contained in the structure and reads the data from the elements having the same name as the corresponding variable. The type of variable must be compatible with the type of data associated with the corresponding element. If the structure contains type STRUCT variables, then the function iterates the read process on this structure. If the structure contains an array of structures then the function iterates the read process on the array of structures; however, if the number of array elements contained in the file (N1) is smaller than the number of elements declared at PLC level (N2), only the first N1 elements are loaded in the array.

Note: variable *VarName* CANNOT be an array element, e.g., ‘ArrName[2]’.

The variable *VarName* can be one of the following:

- A global variable: in this case, just specify the name of the variable in the *VarName* parameter, e.g., ‘GlobalVarName’

- A variable defined within a program: in this case, the name of the program instance followed by the name of the variable must be written in the *VarName* parameter, e.g., ‘ProgramInstanceName.VarName’
- A variable defined within a function block: in this case, you have to write in the *VarName* parameter the name of the program instance followed by the name of the variable must be written in the *VarName* parameter, e.g., ‘ProgramInstanceName.FunctionBlockInstanceName.VarName’

The internal type values in the XML file can be expressed in three different formats:

- if the first character is ‘0’ and the second character is ‘x’ oppure ‘X’, then the number is read as exadecimal. “0x10” is read as 10 exadecimal, that is 16 decimal.
- if the first character is ‘0’ and the second character is not ‘x’ nor ‘X’, then the number is read as an octal integer. For example “021” is read as 21 octal, that is 17 decimal.
- if the first character is included between ‘1’ and ‘9’ then the number is read as a decimal integer. For example, “12” is read as 12 decimal.

Return values:

The following table lists the possible values of variable *ret_code*

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XMLhandle parameter
0x00000005	Memory allocation error
0x0000000A	The element identified by XmlPath does not exist in the XML file
0x0000000C	The type of data contained in the XML file is not compatible with the type of the corresponding PLC variable
0x0000000E	Variable VarName does not exist in the PLC program
0x0000000F	Internal error

Example

```

Ret : dword;
XMLhandle : dint;
IntVar : int;
...
(* opens file SSD\osai\file.xml in read only mode, read the data associated
with element Element and copies it to the IntVar global variable, and finally
closes it *)

Ret := XMLOpen ('\\SSD\osai\file.xml', XML_FMODE_READ, XMLhandle) ;
Ret := XMLRead( XMLhandle, '/top/element', 'IntVar');
Ret := XMLClose(XMLhandle);

...
This is the content of file SSD\osai\file.xml:
<?xml version="1.0"?>
<top>
<element>30</element>
</top>
```

9.4 XMLWrite

Function XMLWrite writes one or more data in the elements of an XML file.

Syntax:

```
ret_code := XMLWrite (XMLhandle, XmlPath, Mode, VarName) ;
```

Input parameters:

<i>XMLhandle</i> (DINT)	univocal identifier of XML file.
<i>XmlPath</i> (STRING)	XML path of the element where the data have to be written
<i>Mode</i> (INT)	write mode
<i>VarName</i> (STRING)	PLC variable from which to read the data to be written in the XML file

Output parameter:

Execution mode:

Wait

Use:

This function writes data in the elements of an XML file; data to be written are read from the *VarName* variable defined within the PLC program.

The XML file where to write the data is identified by parameter *XMLhandle*, this parameter is returned by the XMLOpen function upon the opening of the file.

The *XmlPath* parameter selects an element within the XML file where the data is to be written,, if the element identified by parameter *XmlPath* does not exist in the XML file, then the behaviour of the function depends on the value of the *Mode* parameter.

The following table lists the possible values that can be applied to the *Mode* parameter:

Mode	Mnemonic	Description
10	XML_WMODE_ADD_MISSING	If the element identified by the <i>XmlPath</i> parameter does not exist in the XML file, the function creates it. If non-existent, the entire XML path needed to reach the element is also created.
11	XML_WMODE_SKIP_MISSING	If the element identified by the <i>XmlPath</i> parameter does not exist in the XML file, the function does NOT create it.

Depending on type of *VarName* variable, this function makes it possible to write a single scalar value, or an array of scalar values, or a structure that may even be nested. A structure may also contain arrays of structure, as long as this restriction is complied with: the lowest value of the array index must be 1, e.g., StructArray : ARRAY[1..10] OF Mystruct;

The function searches for the *VarName* function from among the symbols and determines what type it is.

The *VarName* variable type can be one of the following:

- ANY_ELEMENTARY or ARRAY of ANY_ELEMENTARY type. In this case the function reads the data directly from the *VarName* variable and writes them to the XML file.

- **STRUCT.** The function creates a list of the ANY_ELEMENTARY type or ARRAY of ANY_ELEMENTARY type variables contained in the structure and writes the data in the elements having the same name as the corresponding variable. If the structure contains type STRUCT variables, then the function iterates the write process on this structure.

Note: variable *VarName* CANNOT be an array element, e.g., ‘ArrName[2]’.

Variable *VarName* can be one of the following:

- A global variable: in this case, just specify the name of the variable in the *VarName* parameter, e.g., ‘GlobalVarName’
- A variable defined within a program: in this case, the name of the program instance followed by the name of the variable must be written the *VarName* parameter, e.g., ‘ProgramInstanceName.VarName’
- A variable defined within a function block: in this case, you have to write in the *VarName* parameter the name of the program instance followed by the name of the function block instance followed by the name of the variable must be written in the *VarName* parameter, e.g., ‘ProgramInstanceName.FunctionBlockInstanceName.VarName’.

Return values:

The possible values of variable *ret_code* are listed in the following table.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XMLhandle parameter
0x00000004	Invalid Mode parameter
0x00000005	Memory allocation error
0x0000000A	The element identified by the XmlPath parameter does not exist in the XML file
0x0000000B	The XML path cannot be created
0x0000000E	Variable VarName does not exist in the PLC program
0x0000000F	Internal error

Example

```

Ret : dword;
XMLhandle : dint;
IntVar : int := 15;
...
(* opens file SSD\osai\file.xml in write mode, writes the data associated with
element Element reading it from the IntVar global variable, saves the file and
finally closes it *)
Ret := XMLOpen('\SSD\osai\file.xml', XML_FMODE_CREATE, XMLhandle);
Ret := XMLWrite(XMLhandle, '/top/element', XML_WMODE_ADD_MISSING,
'IntVar');
Ret := XMLSave(XMLhandle);
Ret := XMLClose(XMLhandle);
...
The content of the file SSD\osai\file.xml when the previous code has been
executed:
<?xml version="1.0"?>
<top>
<element>15</element>
</top>
```

9.5 XMLReadAttribute

The *XMLReadAttribute* function reads one or more data from the attributes of an XML file.

Syntax:

```
ret_code := XMLReadAttribute (XMLhandle, XmlPath, XmlAttribName, VarName) ;
```

Input parameters:

<i>XMLhandle</i> (DINT)	univocal identifier of XML file.
<i>XmlPath</i> (STRING)	XML path of element where to read the data
<i>XmlAttribName</i> (STRING)	name of XML attribute containing the data
<i>VarName</i> (STRING)	PLC variable where to write the data read from XML file

Output parameters:

Execution mode:

Wait

Use:

This function reads data from the attributes of the elements of an XML file and writes them to a *VarName* variable defined within the PLC program. The XML file where will be read the data is identified by parameter *XMLhandle*, this parameter is returned by the *XMLOpen* function upon the opening of the file. Parameter *XmlPath* selects the element within the XML file from which the data will be obtained, if the element identified by the *XmlPath* parameter does not exist in the XML file, the function returns an error. Parameter *XmlAttribName* selects the attribute where the data is read: this parameter is significant only if *VarName* is a simple variable, not a structure.

Depending on type of *VarName* variable, this function reads a single scalar value or a structure that may even be nested. A structure may even contain arrays of structures provided that this restriction is complied with: the lowest value of the array index must be 1, e.g., *StructArray* : ARRAY[1..10] OF *Mystruct*;

The function searches for the *VarName* function from among the symbols and determines what type it is. The *VarName* variable type can be one of the following:

- › ANY_ELEMENTARY. In this case the function reads the data from attribute *XmlAttribName* of the element identified by the *XmlPath* parameter; the function writes the data directly to variable *VarName*. Type of *VarName* variable must be compatible with type of data associated with attribute *XmlAttribName* of the element identified by the *XmlPath* parameter.
- › STRUCT. The function creates a list of the ANY_ELEMENTARY type variables contained in the structure and reads the data from the attributes of the elements having the same name as the corresponding variable. The type of variable must be compatible with the type of data associated with the corresponding attribute. In this case, parameter *XmlAttribName* is not significant. If the structure contains type STRUCT variables, then the function iterates the read process on this structure. If the structure contains an array of structures then the function iterates the read process on the array of structures; however, if the number of array elements contained in the file (N1) is smaller than the number of elements declared at PLC level (N2), only the first N1 elements are loaded in the array.

Note: variable *VarName* CANNOT be an array element, e.g., ‘ArrName[2]’.

Variable VarName can be one of the following:

- › A global variable: in this case, just specify the name of the variable in the *VarName* parameter, e.g., ‘GlobalVarName’
- A variable defined within a program: in this case, the name of the program instance followed by the name of the variable must be written the *VarName* parameter, e.g., ‘ProgramInstanceName.VarName’
- › A variable defined within a function block: in this case, the name of the program instance followed by the name of the function block instance followed by the name of the variable must be written the *VarName* parameter, e.g., ‘ProgramInstanceName.FunctionBlockInstanceName.VarName’.

The internal type values in the XML file can be expressed in three different formats:

- › if the first character is ‘0’ and the second character is ‘x’ oppure ‘X’, then the number is read as exadecimal. “0x10” is read as 10 exadecimal, that is 16 decimal.
- › if the first character is ‘0’ and the second character is not ‘x’ nor ‘X’, then the number is read as an octal integer. For example “021” is read as 21 octal, that is 17 decimal.
- › if the first character is included between ‘1’ and ‘9’ then the number is read as a decimal integer. For example, “12” is read as 12 decimal.

Return values:

The possible values taken on by variable *ret_code* are listed in the following table:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XMLhandle parameter
0x00000005	Memory allocation error
0x0000000A	The element identified by XmlPath does not exist in the XML file
0x0000000C	Type of data contained in XML file not compatible with that of corresponding PLC variable
0x0000000E	Variable VarName does not exist in the PLC program
0x0000000F	Internal error

Example

```
Ret : dword;
XMLhandle : dint;
IntVar : int;
...
(* opens file SSD\osai\file.xml in read mode, reads the data associated with
attribute attrib1 of the second element elem and copies it to the IntVar global
variable, and finally closes it. XML path '/top/ele[2]' identifies the second
elem and hence, in this example, the value written to the IntVar variable will
be 20. *)
```

```
Ret := XMLOpen ('\SSD\osai\file.xml', XML_FMODE_READ, XMLhandle) ;
Ret := XMLReadAttribute(XMLhandle, '/top/ele[2]', 'attrib1', 'IntVar');
Ret := XMLClose(XMLhandle);
-
```

This is the content of file SSD\osai\file.xml:

```
<?xml version="1.0"?>
<top>
<elem attrib1="10" attrib2="a" />
<elem attrib1="20" attrib2="b" />
<elem attrib1="30" attrib2="c" />
</top>
```

9.6 XMLReadAttributeEx

The XMLReadAttributeEx function reads one or more data from the attributes of an XML file.

Syntax

```
ret_code := XMLReadAttributeEx (XMLhandle, XmlPath, XmlAttribName, Mode, VarName) ;
```

Input parameters

<i>XMLhandle</i> (DINT)	univocal XML file identifier
<i>XmlPath</i> (STRING)	path XMLpath of the element from which reading the data
<i>XmlAttribName</i> (STRING)	XML attribute name containing the data
<i>Mode</i> (WORD)	function execution mode
<i>VarName</i> (STRING)	PLC variable where writing data read by the XML file

Output parameter

Execution mode

Wait

Use

This function lets you read data from the attributes of the elements of an XML file and write them to a *VarName* variable defined within the PLC program. The XML file where to read the data is identified by parameter *XMLhandle*, this parameter is returned by the XMLOpen function upon the opening of the file. Parameter *XmlPath* lets you select the element within the XML file from which to reach the data, if the element identified by the *XmlPath* parameter does not exist in the XML file, the function returns an error. Parameter *XmlAttribName* lets you select the attribute where to read the data: this parameter is significant only if *VarName* is a simple variable, not a structure.

Depending on type of *VarName* variable, this function lets you read a single scalar value or a structure that may even be nested. A structure may even contain arrays of structures provided that this restriction is complied with: the lowest value of the array index must be 1, e.g., StructArray : ARRAY[1..10] OF Mystruct;

The function searches for the *VarName* function from among the symbols and determines what type it is. The *VarName* variable type can be one of the following:

- ANY_ELEMENTARY. In this case the function reads the data from attribute *XmlAttribName* of the element identified by the *XmlPath* parameter; the function writes the data directly to variable *VarName*. Type of *VarName* variable must be compatible with type of data associated with attribute *XmlAttribName* of the element identified by the *XmlPath* parameter.
- STRUCT. The function creates a list of the ANY_ELEMENTARY type variables contained in the structure and reads the data from the attributes of the elements having the same name as the corresponding variable. The type of variable must be compatible with the type of data associated with the corresponding attribute. In this case, parameter *XmlAttribName* is not significant. If the structure contains type STRUCT variables, then the function iterates the read process on this structure. If the structure contains an array of structures then the function iterates the read process on the array of structures; however, if the number of array elements contained in the file (N1) is smaller than the number of elements declared at PLC level (N2), only the first N1 elements are loaded in the array.

Note: variable *VarName* CANNOT be an array element, e.g., ‘ArrName[2]’.

Variable *VarName* can be one of the following:

- › A global variable: in this case, just specify the name of the variable in the *VarName* parameter, e.g., ‘GlobalVarName’
- › A variable defined within a program: in this case, you have to write in the *VarName* parameter the name of the program instance followed by the name of the variable, e.g., ‘ProgramInstanceName.VarName’
- › A variable defined within a function block: in this case, you have to write in the *VarName* parameter the name of the program instance followed by the name of the function block instance followed by the name of the variable, e.g., ‘ProgramInstanceName.FunctionBlockInstanceName.VarName’.

Mode parameter is a bits mask with the following meaning:

Bit	Description and possible values
0	If set =1: if the XML file is missing one or more attributes the user wants to read, then the function successfully returns If set =0: if the XML file is missing one or more attributes the user wants to read, the function returns error
1..15	Not used (set =0)

The internal type values in the XML file can be expressed in three different formats:

- › if the first character is ‘0’ and the second character is ‘x’ oppure ‘X’, then the number is read as exadecimal. “0x10” is read as 10 exadecimal, that is 16 decimal.
- › if the first character is ‘0’ and the second character is not ‘x’ nor ‘X’, then the number is read as an octal integer. For example “021” is read as 21 octal, that is 17 decimal.
- › if the first character is included between ‘1’ and ‘9’ then the number is read as a decimal integer. For example, “12” is read as 12 decimal.

Return values

The following table lists all possible values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00380002	Invalid XMLhandle parameter
0x00380005	Memory allocation error
0x0038000A	The element identified by XmlPath does not exist in the XML file
0x0038000C	Type of data contained in XML file not compatible with that of corresponding PLC variable
0x0038000E	Variable VarName does not exist in the PLC program
0x0038000F	Internal error
0x00380011	Attribute empty or not present in the XML file

Example

```
Ret      : dword;
XMLhandle : dint;
IntVar    : int;
...
(* opens the SSD\osai\file.xml file in read only mode, reads data referred to
the attrib1 attribute of the second elem element, copying it in the IntVar
global variable, and then closes it. The XML '/top/elem[2]' path identifies the
second elem, therefore in this example value 20 will be written in the IntVar
variable. *)

Ret := XMLOpen ('\SSD\osai\file.xml', XML_FMODE_READ, XMLhandle) ;
Ret := XMLReadAttributeEx(XMLhandle,'/top/elem[2]', 'attrib1', 0, 'IntVar') ;
Ret := XMLClose(XMLhandle);

-
This is the content of the SSD\osai\file.xml file:
<?xml version="1.0"?>
<top>
  <elem attrib1="10" attrib2="a" />
  <elem attrib1="20" attrib2="b" />
  <elem attrib1="30" attrib2="c" />
</top>
```

9.7 XMLReadEx

The XMLReadEx function reads one or more data from XML file elements.

Syntax

```
ret_code := XMLReadEx (XMLhandle, XmlPath, Mode, VarName) ;
```

Input parameters

<i>XMLhandle</i> (DINT)	Univocal XML identifier
<i>XmlPath</i> (STRING)	path XML path of the element form which reading the data
<i>Mode</i> (WORD)	Function execution mode
<i>VarName</i> (STRING)	PLC variable where writing data read by XML file

Output parameter

Execution mode

Wait

Use:

This function reads data from the elements of an XML file and write them in a variable, called *VarName*, defined in the PLC program.

The XML file from which the data are to be read is identified by parameter *XMLhandle*, this parameter is returned by the XMLOpen function upon the opening of the file.

Parameter *XmlPath* selects an element in the XML file from which the data are to be read; if the element identified by the *XmlPath* parameter does not exist in the XML file, then the function returns an error.

Depending on the type of *VarName* variable, this function lets you read a single scalar value or an array of scalar values, or a structure that may even be nested. A structure may even contain arrays of structures, provided that the following restriction is complied with: the lowest value of the array index must be 1, e.g., StructArray: ARRAY[1..10] OF Mystruct;

The function searches for the *VarName* function from among the symbols and determines what type it is. The *VarName* variable type can be one of the following:

- ANY_ELEMENTARY or ARRAY of ANY_ELEMENTARY type. In this case the function reads the data from the XML file and writes the data directly to variable *VarName*. The type of *VarName* variable must be compatible with the type of data associated with the element identified by parameter *XmlPath*.
- STRUCT. The function creates a list of the ANY_ELEMENTARY type or ARRAY of ANY_ELEMENTARY type variables contained in the structure and reads the data forform the elements having the same name as the corresponding variable. The type of variable must be compatible with the type of data associated with the corresponding element. If the structure contains type STRUCT variables, then the function iterates the read process on this structure. If the structure contains an array of structures then the function iterates the read process on the array of structures; however, if the number of array elements contained in the file (N1) is smaller than the number of elements declared at PLC level (N2), only the first N1 elements are loaded in the array.

Note: variable *VarName* CANNOT be an array element, e.g., ‘ArrName[2]’.

The variable *VarName* can be one of the following:

- A global variable: in this case, just specify the name of the variable in the *VarName* parameter, e.g., ‘GlobalVarName’
- A variable defined within a program: in this case, you have to write in the *VarName* parameter the name of the program instance followed by the name of the variable, e.g., ‘ProgramInstanceName.VarName’
- A variable defined within a function block: in this case, you have to write in the *VarName* parameter the name of the program instance followed by the name of the function block instance followed by the name of the variable, e.g., ‘ProgramInstanceName.FunctionBlockInstanceName.VarName’

The *Mode* parameter is a bit mask having the following meaning:

Bit	Description and possible values
0	set =1: if the XML file is missing one or more attributes the user wants to read, then the function successfully returns If set =0: if the XML file is missing one or more attributes the user wants to read, the function returns error
.15	Not used (set =0)

The internal type values in the XML file can be expressed in three different formats:

- if the first character is ‘0’ and the second character is ‘x’ oppure ‘X’, then the number is read as exadecimal. “0x10” is read as 10 exadecimal, that is 16 decimal.
- if the first character is ‘0’ and the second character is not ‘x’ nor ‘X’, then the number is read as an octal integer. For example “021” is read as 21 octal, that is 17 decimal.
- if the first character is included between ‘1’ and ‘9’ then the number is read as a decimal integer. For example, “12” is read as 12 decimal.

Return values

The following table lists all possible values that *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XMLhandle parameter
0x00000005	Memory allocation error
0x0000000A	The element identified by XmlPath does not exist in the XML file
0x0000000C	The type of data contained in the XML file is not compatible with the type of the corresponding PLC variable
0x0000000E	Variable VarName does not exist in the PLC program
0x0000000F	Internal error

Example

```
Ret      : dword;
XMLhandle : dint;
IntVar    : int;
...
(* open in read mode the file SSD\osai\file.xml, reads the data referred to the
element element copying it in the IntVar global variable, then closes it *)

Ret := XMLOpen ('\SSD\osai\file.xml', XML_FMODE_READ, 0, XMLhandle) ;
Ret := XMLReadEx( XMLhandle, '/top/element', 'IntVar') ;
Ret := XMLClose(XMLhandle);

...
This is the content of SSD\osai\file.xml file:
<?xml version="1.0"?>
<top>
  <element>30</element>
</top>
```

9.8 XMLWriteAttribute

Function XMLWriteAttribute writes one or more data in the attributes of an XML file

Syntax:

```
ret_code := XMLWriteAttribute (XMLhandle, XmlPath, XmlAttributeName, Mode, VarName) ;
```

Input parameters:

<i>XMLhandle</i> (DINT)	univocal identifier of XML file.
<i>XmlPath</i> (STRING)	XML path of element where to write the data
<i>XmlAttributeName</i> (STRING)	name of attribute where to write the data
<i>Mode</i> (INT)	write mode
<i>VarName</i> (STRING)	PLC variable from which reading the data to write in the XML file

Output parameters:

Execution mode:

Wait

Use:

This function writes data to the attributes of an XML file; the data to be written will be read by the *VarName* variable defined within the PLC program.

The XML file where the data will be written is identified by parameter *XMLhandle*, this parameter is returned by the XMLOpen function upon the opening of the file.

Parameter *XmlPath* selects an element within the XML file where the data will be written.

Parameter *XmlAttributeName* selects an attribute where the data will be written; this parameter is significant only if *VarName* is a simple variable, as opposed to a structure.

If the element identified by the *XmlPath* parameter does not exist in XML file, or attribute *XmlAttributeName* does not exist in the XML file, then the behaviour of the function depends on the value of the *Mode* parameter. The following table lists the possible values of the *Mode* parameter

Mode	Mnemonic	Description
10	XML_WMODE_ADD_MISSING	If the element identified by the <i>XmlPath</i> parameter does not exist in XML file, then the function creates it. If non-existent, the entire XML path needed to reach the element is also created. If the <i>XmlAttributeName</i> attribute does not exist in XML file, then the function creates it.
11	XML_WMODE_SKIP_MISSING	If the element identified by the <i>XmlPath</i> parameter does not exist in XML file, then the function does NOT create it. If the <i>XmlAttributeName</i> attribute does not exist in the XML file, the function does NOT create it.

Depending on the type of *VarName* variable, this function lets you write a single scalar value or a structure, which may even be nested. A structure may even contain arrays of structures provided that this restriction is complied with: the lowest value of the array index must be 1, e.g., StructArray: ARRAY[1..10] OF Mystruct;

The function searches for the *VarName* function from among the symbols and determines what type it is.

The *VarName* variable type can be one of the following:

- ANY_ELEMENTARY. In this case, the function reads the data directly from the *VarName* variable and writes them to the XML file, in the *XmlAttributeName* attribute of the element identified by *XmlPath*
- STRUCT. The function creates a list of type ANY_ELEMENTARY variables contained in the structure and writes the data in the attributes having the same name as the corresponding variable. If the structure contains type STRUCT variables, then the function iterates the write process on this structure. In this case, parameter *XmlAttributeName* is not significant.

Note: variable *VarName* CANNOT be an array element, e.g., ‘ArrName[2]’.

Variable *VarName* can be one of the following:

- A global variable: in this case, just specify the name of the variable in the *VarName* parameter, e.g., ‘GlobalVarName’
- A variable defined within a program: in this case, you have to write in the *VarName* parameter the name of the program instance followed by the name of the variable, e.g., ‘ProgramInstanceName.VarName’
- A variable defined within a function block: in this case, you have to write in the *VarName* parameter the name of the program instance followed by the name of the function block instance followed by the name of the variable, e.g., ‘ProgramInstanceName.FunctionBlockInstanceName.VarName’.

Return values:

The possible values of variable *ret_code* are listed in the following table:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XMLhandle parameter
0x00000004	Invalid Mode parameter
0x00000005	Memory allocation error
0x0000000A	The element identified by <i>XmlPath</i> does not exist in the XML file
0x0000000B	XML path cannot be created
0x0000000E	Variable <i>VarName</i> does not exist in the PLC program
0x0000000F	Internal error

Example

```
Ret : dword;
XMLhandle : dint;
IntVar : int := 15;
...
(* opens file SSD\osai\file.xml in write mode, writes the data associated with
the attribute attrib of the element elem reading it from the IntVar global
variable, saves the file and finally closes it *)

Ret := XMLOpen('\SSD\osai\file.xml', XML_FMODE_CREATE, XMLhandle);
Ret := XMLWriteAttribute(XMLhandle, '/top/elel', 'attrib',
    XML_WMODE_ADD_MISSING, 'IntVar');
Ret := XMLSave(XMLhandle);
Ret := XMLClose(XMLhandle);
...
This is the content of file SSD\osai\file.xml when the previous code has been
executed:
<?xml version="1.0"?>
<top>
  <elel attrib="15" />
</top>
```

9.9 XMLRemove

Function XMLRemove removes an element from an XML file.

Syntax:

```
ret_code := XMLRemove (XMLhandle, XmlPath) ;
```

Input parameters:

<i>XMLhandle(DINT)</i>	Univocal identifier of XML file
<i>XmlPath (STRING)</i>	XML path of the element to be removed

Execution mode:

Wait

Use:

This function lets you remove an element from an XML file. The XML file with the element to be removed is identified by parameter *XMLhandle*, this parameter is returned by the XMLOpen function upon the opening of the file. Parameter *XmlPath* selects the element to be removed within the XML file. If under the chosen element there is a sub-tree of elements, the entire sub-tree is removed too.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XML handle parameter
0x0000000A	The element identified by XmlPath does not exist in XML file

Example:

```
Ret : dword;
XMLhandle : dint;
...
(* opens file SSD\osai\file.xml in read/write mode, removes the element elem2,
saves the file and finally closes it *)
Ret := XMLOpen ('\SSD\osai\file.xml', XML_FMODE_READ_WRITE, XMLhandle);
Ret := XMLRemove (XMLhandle, '/top/elem2');
Ret := XMLSave (XMLhandle);
Ret := XMLClose (XMLhandle);
...
This is the content of file SSD\osai\file.xml before the execution of
XMLRemove:
<?xml version="1.0"?>
<top>
<elem1>10</elem1>
<elem2>11</elem2>
</top>
```

This is the content of file SSD\osai\file.xml after the execution of XMLRemove:

```
<?xml version="1.0"?>
<top>
  <elem1>10</elem1>
</top>
```

9.10 XMLRemoveAttribute

Function XMLRemoveAttribute removes an attribute from an XML file.

Syntax:

```
ret_code := XMLRemoveAttribute (XMLhandle, XmlPath, XmlAttribName) ;
```

Input parameters:

<i>XMLhandle</i> (DINT)	Univocal identifier of XML file
<i>XmlPath</i> (STRING)	XML path of the element
<i>XmlAttribName</i> (STRING)	Name of attribute to be removed

Execution mode:

Wait

Use:

This function lets you remove an attribute from an XML file. The XML file with the attribute to be removed is identified by parameter *XMLhandle*, this parameter is returned by the XMLOpen function upon the opening of the file. Parameter *XmlPath* lets you select an element within the XML file.

Parameter *XmlAttribName* specifies the name of the attribute to be removed; this attribute is associated with the element selected with *XmlPath*.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XMLhandle parameter
0x0000000A	The element identified by XmlPath does not exist in the XML file

Example:

```
Ret : dword;
XMLhandle : dint;
...
(* opens file SSD\osai\file.xml in read/write mode, removes the attribute
attrib1 associated with the element elem, saves the file and finally closes it
*)

Ret := XMLOpen(`\SSD\osai\file.xml', XML_FMODE_READ_WRITE, XMLhandle);
Ret := XMLRemoveAttribute(XMLhandle, '/top/elem', 'attrib1');
Ret := XMLSave(XMLhandle);
Ret := XMLClose(XMLhandle);
...
This is the content of file SSD\osai\file.xml before the execution of
XMLRemoveAttribute:
<?xml version="1.0"?>
<top>
<elem attrib1="1.1" attrib2="2.1" />
<elem2 attrib3="3.0" attrib4="4.0" />
</top>
```

This is the content of file SSD\osai\file.xml following the execution of XMLRemoveAttribute:

```
<?xml version="1.0"?>
<top>
<elem attrib2="2.1" />
<elem2 attrib3="3.0" attrib4="4.0" />
</top>
```

9.11 XMLSave

Function XMLSave saves an XML file in file system.

Syntax:

```
ret_code := XMLSave (XMLhandle) ;
```

Input parameters:

<i>XMLhandle</i> (DINT)	Handle to an already opened XML file
-------------------------	--------------------------------------

Execution mode:

Wait

Use:

This function lets you save an XML file in file system.

The XML file to be saved is identified by parameter *XMLhandle*, this parameter is returned by the XMLOpen function upon the opening of the file.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00000002	Invalid XML handle parameter
0x00000008	File cannot be saved having been opened in read only mode
0x00000009	File cannot be saved

Example:

```
Ret : dword;
XMLhandle : dint;
IntVar : int;
...
(* opens file SSD\osai\file.xml in write mode, writes to it creating a top
element with attribute, saves the file in file system with function XMLSave,
and finally closes it *)
Ret := XMLOpen ('\SSD\osai\file.xml', XML_FMODE_CREATE, XMLhandle) ;
Ret := XMLWriteAttribute( XMLhandle, '/top', 'attrib',
    XML_WMODE_ADD_MISSING, 'IntVar') ;
Ret := XMLSave(XMLhandle) ;
Ret := XMLClose(XMLhandle) ;
```

10. Generic system functions - 4C.Utility2

Loadlibrary	Loads a dll written by the user into the address space of the calling process
Freelibrary	Downloads a dll written by the user from the address space of the calling process
Runprogram	Allows to launch a program
Getdate	Reads the system date
Gettime	Reads the system time
Setdate	Allows to change the system date
Settime	Writes the system time
Closedb	Closes a database containing PLC data structures
CpuUsage	Calculates the CPU usage percentage
Deletedb	Deletes a database containing PLC data structures
ForceRemapInput	Defines a physical digital input remapping on a logic digital input
OpenDB	Opens a database containing PLC data structures
ReadDB	Reads database record containing PLC data structures
ReadRemapOutputCFG	Uploads in memory a file of digital inputs remapping definitions
AddRemapOutputCFG	Uploads in memory a file of output remap definitions, without deleting the definitions previously uploaded.
RemapOutput	Executes outputs remapping
RemapOutputByClass	Remaps the outputs according to their belonging class
DeleteRemapOutput	Deletes from memory all the remap definitions of the outputs
ResizeDB	Changes the number of records in a database containing PLC data structures
CopyDB	Copies a database containing PLC data structures
CreateDB	Creates database to store PLC data structures
FilelistDB	Provides database list
ForceRemapOutput	Allows to define a physical digital output remapping on a logic digital output
PurgeRetainDB	Restores the retentive memory covering spaces no more in use
ReadRemapInputCFG	Allows to upload in memory a file of digital inputs remapping definitions
AddRemapInputCFG	Allows to upload in memory a file of inputs remap definitions, without deleting the definition already uploaded
RemapInput	Executes input remapping
RemapInputByClass	Runs the inputs remapping according to their belonging class
DeleteRemapInput	Deletes from memory all the inputs remap definitions
GetRemapAddress	Determines the remapping address of a physical or logical input/output
RenameDB	Renames the database containing PLC data structures
WriteDB	Writes a database record containing PLC data structures

10.1 LoadLibrary

The LoadLibrary function loads a dynamic link library (DLL) written by the user into the address space of the calling process.

Syntax:

```
ret_code := LoadLibrary(LibraryName, StartFunctionOrd, EndFunctionOrd, StartFBOrd, EndFBOrd);
```

Input parameters:

<i>LibraryName</i> (STRING)	Name of DLL to be loaded, with its entire path
<i>StartFunctionOrd</i> (INT)	Ordinal of the library function with the smallest ordinal
<i>EndFunctionOrd</i> (INT)	Ordinal of the library function with the biggest ordinal
<i>StartFBOrd</i> (INT)	Ordinal of the library function block with the smallest ordinal
<i>EndFBOrd</i> (INT)	Ordinal of the library function block with the biggest ordinal

Use:

This function is used to load a DLL written by the user into the address space of the calling process; once DLL has been loaded, the PLC application can use the functions and the function blocks contained in that DLL.

The *LibraryName* parameter contains the name of the DLL to be loaded and the relevant path.

Each function belonging to a DLL must be assigned an univocal ordinal number which must correspond to the ordinal assigned to the function in the .def file of the DLL.

The ordinal numbers of the functions contained in the DLL must fall within the range specified by parameters *StartFunctionOrd* and *EndFunctionOrd*.

The ordinals of the library functions must in any case lie in the 1000 - 1499 range.

If a library contains no functions, parameters *StartFunctionOrd* and *EndFunctionOrd* must both be assigned the value of 0.

Each function block belonging to a DLL must be assigned a univocal ordinal number which must correspond to the ordinal assigned to the function block in the .def file of the DLL.

The ordinal numbers of the function blocks contained in the DLL must fall within the range specified by parameters *StartFBOrd* and *EndFBOrd*.

The ordinals of the library function blocks must in any case lie in the 500 - 999 range.

If a library contains no function blocks, parameters *StartFBOrd* and *EndFBOrd* must both be assigned the value of 0.

It is possible to define within the DLL a UserInitLibrary function that initialises the DLL: the UserInitLibrary function must have ordinal 1 in the .def file of the DLL. The LoadLibrary function determines whether or not there is a function with ordinal 1 in the DLL and, if there is, calls it. The UserInitLibrary function must have the following prototype:

```
BOOL UserInitLibrary(dword *pErr);
```

If the process fails, the UserInitLibrary returns FALSE (0); in this case, the error code returned by the UserInitLibrary in the pErr parameter is returned in its turn, as a return value of the LoadLibrary function.

It is possible to define within the DLL a UserTerminateLibrary that frees all the resources, if any, taken up by the DLL: the UserTerminateLibrary function must have ordinal 2 in the .def file of the DLL. The LoadLibrary function determines whether or not there is a function with ordinal 2 in the DLL; if there is, the UserTerminateLibrary function will be called if either the following circumstances occurs:

- ▷ The FreeLibrary function that downloads the DLL is called.
- ▷ A new PLC application is downloaded onto a target

The UserTerminateLibrary function must have the following prototype:

```
BOOL UserTerminateLibrary();
```

Several DLL users can be loaded; the maximum number of DLLs that can be loaded simultaneously is 8. However, there is a condition to be complied with: the ranges of the ordinals of the different DLLs user must never overlap. A similar restriction also applies to the ranges of the ordinals of the function blocks.

Return values:

Value (HEX)	Description
0	Function executed successfully
0x0015009B	Parameter <i>StartFunctionOrd</i> or parameter <i>EndFunctionOrd</i> outside the 1000 - 1499 range
0x0015009C	Parameter <i>StartFBOrd</i> or parameter <i>EndFBOrd</i> outside the 500 - 999 range
0x0015009D	This DLL has already been loaded
0x0015009E	The range of the function ordinals overlaps the range of a previously loaded DLL.
0x0015009F	The range of the function block ordinals overlaps the range of a previously loaded user DLL.
0x001500A0	Error in loading DLL
0x001500A1	The maximum admissible number of DLLs loaded simultaneously has been exceeded
User defined errors	If the UserInitLibrary function is present and returns error, this error is included in the LoadLibrary return value.

Example:

In this example, DLL '\SSD\UserDll.dll' is loaded.

The DLL functions have ordinals in the 1010 - 1020 range; the DLL does not contain any function blocks.

VAR

```
RetCode : dword;
END_VAR
```

```
RetCode := LoadLibrary('\SSD\UserDll2.dll', 1010, 1020, 0, 0);
```

See also: FreeLibrary

10.2 FreeLibrary

The FreeLibrary function downloads a user DLL from the address space of the calling process.

Syntax:

```
ret_code := FreeLibrary(LibraryName);
```

Input parameters:

<i>LibraryName</i> (STRING)	Name of DLL to be loaded with its entire path
-----------------------------	---

Use:

This function makes it possible to download a DLL written by the user from the address space of the calling process; once the DLL has been downloaded, the PLC application will no longer be able to use the functions and the function blocks contained in the DLL.

Parameter *LibraryName* contains the name, with the path, of the DLL to be downloaded.

If the .def file of the DLL contains a function with ordinal 2, this function is called within the FreeLibrary. This user function (UserTerminateLibrary) can be used to free all the resources, if any, taken up by the DLL.

For the FreeLibrary to be executed successfully, the user DLL must have been previously loaded through the LoadLibrary function.

Return values:

Value (HEX)	Description
0	Function executed successfully
0x001500A2	Error in downloading the DLL
0x001500A3	The DLL to be downloaded has not been previously loaded

Example:

In this example, DLL '\SSD\UserDll.dll' is downloaded.

```
VAR
  RetCode : dword;
END_VAR
RetCode := FreeLibrary('\SSD\UserDll2.dll');
```

See also:

[LoadLibrary](#)

10.3 RunProgram

The RunProgram function launches a program.

Syntax:

```
ret_code := RunProgram (ExecMode, ProgramName, CommandLine, Timeout);
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>ProgramName</i> (STRING)	Full name of program to be launched
<i>CommandLine</i> (STRING)	Possible command line to be input to program
<i>Timeout</i> (UDINT)	Timeout, expressed in ms

Use:

This function launches a program, possibly by entering parameters into the command line.

Parameter *ProgramName* contains the name of the program to be launched with the relevant path,.

If parameter *ExecMode* is MODE_NOWAIT, then the function launches the program and returns immediately; in this case the *Timeout* parameter is not significant.

On the other hand, if parameter *ExecMode* is MODE_WAIT, then the function waits a predetermined period of time for the program just launched to finish running; this period of time corresponds to the value of the *Timeout* parameter.

The *Timeout* parameter is expressed in ms. If parameter *Timeout* is 0, the wait time will be infinite.

Return values:

Value (HEX)	Description
0	Function executed successfully
0x001500A8	Error in launching the program
0x001500A9	Program launched failed to end within timeout

Example:

In this example, the function launches executable \ssd\test_exe.exe in mode MODE_NOWAIT, without entering any parameter with the command line.

```
VAR
  Ret : dword;
END_VAR
Ret := RunProgram(MODE_NOWAIT, '\ssd\test_exe.exe', '', 0);
```

10.4 GetDate

The GetDate function reads the system date.

Syntax:

```
ret_code := GetDate (Day, Month, Year, DayOfWeek) ;
```

Output parameters:

Day (ANY_ELEMENTARY)	Day
Month (ANY_ELEMENTARY)	Month
Year (ANY_ELEMENTARY)	Year
DayOfWeek (ANY_ELEMENTARY)	Day of the week

Possible overloads:

GetDate(INT,INT,INT,INT)
GetDate(WORD,WORD,WORD,WORD)

Execution modality:

Immediate

Use:

Returns the system date currently set in the system. As for the day of the week, 0 stands for Sunday, 1 for Monday...and so on.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Day : INT ;
Month : INT ;
Year : INT ;
DayOfWeek : INT ;
Ret := GetDate (Day, Month, Year, DayOfWeek) ;
```

See also:

[SetDate](#), [GetTime](#), [SetTime](#)

10.5 GetTime

The GetTime function reads the system time.

Syntax:

```
ret_code := GetTime (Hour, Minute, Second) ;
```

Output parameters:

<i>Hour</i> (ANY_ELEMENTARY)	Hour
<i>Minute</i> (ANY_ELEMENTARY)	Minute
<i>Second</i> (ANY_ELEMENTARY)	Second

Possible overloads:

GetTime(INT,INT,INT)

GetTime(WORD,WORD,WORD)

Execution modality:

Immediate

Use:

The GetTime function is used to read the system time.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors

Example:

```
Ret : dword;
Hour : INT ;
Minute : INT ;
Second : INT ;
Ret := GetTime (Hour, Minute, Second) ;
```

See also:

[GetDate](#), [SetDate](#), [SetTime](#)

10.6 SetDate

The SetDate function enables the system date to be changed.

Syntax:

```
ret_code := SetDate (Day, Month, Year) ;
```

Input parameters:

Day (ANY_ELEMENTARY)	Day
Month (ANY_ELEMENTARY)	Month
Year (ANY_ELEMENTARY)	Year

Possible overloads:

SetDate(INT,INT,INT)
SetDate(WORD,WORD,WORD)

Execution modality:

Immediate

Use:

The SetDate function is used to change the system date.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0015000E	Wrong date

Example:

```
Ret : dword;  
Ret := SetDate (23, 8, 1960) ;
```

See also:

GetDate, SetTime, GetTime

10.7 SetTime

The SetTime writes the system time

Syntax:

```
ret_code := SetTime (Hour, Minute, Second) ;
```

Input parameters:

<i>Hour</i> (ANY_ELEMENTARY)	Hour
<i>Minute</i> (ANY_ELEMENTARY)	Minute
<i>Second</i> (ANY_ELEMENTARY)	Second

Possible overloads:

SetTime(INT,INT,INT)
SetTime(WORD,WORD,WORD)

Execution modality:

Immediate

Use:

The SetTime function is used to change the system time.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x0015000D	Time wrong

Example:

```
Ret : dword;  
Ret := SetTime ( 6, 33, 0) ;
```

See also:

GetTime, SetDate, GetDate

10.8 CloseDB

The CloseDB function closes a database containing PLC data structures

Syntax

```
ret_code := CloseDB (DbHandle, DbType) ;
```

Input parameters

<i>DbHandle</i> (DINT)	handle to the database
<i>DbType</i> (INT)	retentive memory or memory mapped file

Execution mode

Wait

Use

The function closes a database containing PLC data structures.

The database to close is identified by the handle *DbHandle* parameter (OpenDB output parameter), and by the database storage on a retentive memory or on a memory mapped file (*DbType* parameter).

For a successful database closing, these conditions are mandatory:

- ▷ The database to close must be present, therefore it should have been previously created using the CreateDB function
- ▷ The database to close has to be opened, therefore it should have been previously opened using the OpenDB function, returning the handle that identifies the database.

There are some operations (for instance the database renaming) that cannot run if a database is opened.

The following table lists all possible values that *DbType* parameter can have:

DbType	Mnemonic	Description
0	dbRetainMemory	Database to be removed is stored in the retentive memory
1	dbMemoryMappedFile	Database to be removed is stored in the memory mapped file

Return values

The following table lists all possible values that *ret_code* parameter can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0008	<i>DbType</i> error value
0x003C0011	Database cannot be closed as it is not opened

Example

```
Ret      : dword;
DbHandle : dint;
...
(* closes the database identified by DbHandle handle, stored on the memory
mapped file *)
Ret := CloseDB(DbHandle, dbMemoryMappedFile);
```

See also: CreateDB, OpenDB, ReadDB, WriteDB

10.9 CpuUsage

The CpuUsage function calculates the CPU usage percentage.

Syntax

```
ret_code := CpuUsage (CpuUsePercentage) ;
```

Output parameter

CpuUsePercentage (UINT)	CPU usage percentage
-------------------------	----------------------

Use

The CpuUsage function calculates the CPU usage percentage returning this value in the *CpuUsePercentage* output parameter.

For instance, if the CPU usage percentage is 99, it means that CPU is almost completely busy.

Return value

Value (HEX)	Description
0	Function executed without errors

Example

```
VAR
res: DWORD ;
CpuUsePercentage : UINT ;

END_VAR

res := CpuUsage (CpuUsePercentage);
```

10.10 DeleteDB

The DeleteDB function deletes a database containing PLC data structures.

Syntax

```
ret_code := DeleteDB (DbName, DbType) ;
```

Input parameters

<i>DbName</i> (STRING)	database name to be removed
<i>DbType</i> (INT)	retentive memory or memory mapped file

Execution mode

Wait

Use

The function deletes a database containing PLC data structures.

The database to delete is identified by name (defines on the *DbName* parameter) and by the database storage on a retentive memory or on a memory mapped file (*DbType* parameter).

The following table lists all possible values that *DbType* parameter can have:

<i>DbType</i>	<i>Mnemonic</i>	Description
0	dbRetainMemory	Database to be removed is stored in the retentive memory
1	dbMemoryMappedFile	Database to be removed is stored in the memory mapped file

The database to be removed has to be previously created using the CreateDB function.

Return values

The following table lists all possible values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0002	Data base name error; usually the name is too long
0x003C0005	Internal error
0x003C0008	<i>DbType</i> value error
0x003C0009	Database cannot be removed as it is opened (see OpenDB function)
0x003C000A	Database does not exist: it cannot be deleted

Example

```
Ret      : dword;
...
(* removes the Axes database stored in memory mapped file *)
Ret := DeleteDB('Axes', dbMemoryMappedFile);
```

See also

CreateDB, OpenDB

10.11 ForceRemapInput

The ForceRemapInput function remaps a physical input on a logical input.

Syntax

```
ret_code := ForceRemapInput (RemapStruct) ;
```

Input parameters

RemapStruct (ANY) structure defining the digital input remap

Possible overloads

ForceRemapInput (InRemapStructType)

ForceRemapInput (InExtraRemapStructType)

Execution mode

Wait

Use

The ForceRemapInput function allows to define a physical input (I variable bit) remapping on a logical input (M variable). The input types below can be remapped:

bool: a physical digital input is remapped on a logical digital input

byte: a physical byte input is remapped on a logical byte input

word: a physical word input is remapped on a logical word input

dword: a physical dword input is remapped on a logical dword input

The function uploads in memory the input remapping definition:

- ▷ If in the memory there is already a remapping definition referred to the same logic input (due to previous calls to the ReadRemapInputCFG function or to the ForceRemapInput function) this definition is overwritten by the ForceRemapInput.
- ▷ If the memory does not have any definition referred to the same logical input, the definition is created and it is added to definitions already existing.

Note that ForceRemapInput uploads in memory only the input remapping definition, without actually running the remapping. For a real remapping execution, the function RemapInput() that executes the inputs remap, has to be called.

Each element can be associated to a message identifier *IOMsgID* (used to view a message in the tool of the remapping/forcing view) and to an *IOClass* used to execute the RemapInput function only for the inputs belonging to a given class. Note that the function DOES NOT check if the different remapping definitions could write the same logical input area; for instance, if a definition writes the MW10 word and another definition writes the MB20 byte, the function does not display any error, even if the two writings conflict.

Each remapping definition has a mode (*Mode*) specifying how to execute the remapping. The existing modes are:

- ▷ Ignore: the logical input does not change
- ▷ Copy: the same physical input value is assigned to the logical input
- ▷ SetToTrue: true value is assigned to the logical input, apart from the physical input value. This mode can be applied only to digital inputs.

- ▷ SetToFalse: false value is assigned to the logical input, apart from the physical input value. This mode can be applied on the digital inputs.
- ▷ Negate: the negative value of the physical input is assigned to the logical input, bit by bit
- ▷ SetToValue: a fixed value is assigned to the logical input, apart from the physical input value.

Two filters can be defined on the digital input:

- ▷ The rise filter is characterized by RiseTime parameter expressed in micro-seconds. The rise filter is defined only if RiseTime > 0.
- ▷ The fall filter is characterized by the FallTime parameter, expressed in micro-seconds. The fall filter is defined only if FallTime > 0.

These filters can be used only in Negate and Copy modes.

In the Copy mode the filters behave as follow:

- ▷ if the rise filter is defined, after the physical input shift from false to true, the logical input shifts at true only if the physical input remains at true for a period equal -at least- to RiseTime
- ▷ if the fall filter is defined, after the physical input shift from true to false, the logical input shifts at false only if the physical input remains at false for a period equal -at least- to FallTime

In the Negate mode the filters behave as follow:

- ▷ if the rise filter is defined, after the physical input shift from true to false, the logical input shifts at true only if the physical input remains at false for a period equal -at least- to RiseTime
- ▷ if the fall filter is defined, after the physical input shift from false to true, the logical input shifts at false only if the physical input remains at true for a period equal -at least- to FallTime

Filters cannot be defined on other inputs as byte, word, dword.

The remapping definition can be made passing to the function one of the structures below:

- ▷ *InExtraRemapStructType*. This structure can be used only starting from OPENcontrol rel. 3.1.
- ▷ *InRemapStructType*. This structure can be used also for releases previous to 3.1, but with limitations in using the *InExtraRemapStructType* structure and in kept only to comply with previous versions. The limitations are:
 1. Only digital inputs are managed (Boolean type); byte, word, dword input types are not managed
 2. Filters on digital inputs are not managed
 3. The SetToValue mode is not managed

The *InExtraRemapStructType* structure contains the remapping definition of an input; this structure is made by the following fields:

Field	Type	Description
VarType	BYTE	Defines the input type to be remapped: bool, byte, word, dword
PhysAddr	UINT	Specifies in bytes the offset in the physical inputs area. i.e. for a byte type physical input the value 13 specifies the IB113 byte
PhysBit	BYTE	For a bool type input, it specifies a bit within the physical input byte defined in the PhysAddr field. Can have value from 0 to 7.
LogicAddr	UINT	Specifies the offset in byte in the logical inputs area. i.e. for a word type logical input, the value 128 specifies the MW64 word.
LogicBit	BYTE	For a bool type input, it specifies a bit in the logical input byte defined in the LogicAddr field. Can have values form 0 to 7.
Mode	BYTE	Specifies the remapping execution mode
Value	UDINT	It is important only in SetToValue mode. It specifies the value assigned to the logical input.

Field	Type	Description
RiseTime	UDINT	Important only for digital inputs and only in the Cpy and Negate modes. Specifies the time of the rise filter, in micro-seconds. The rise filter is defined only if Rise Tme >0.
FallTime	UDINT	Important only for digital inputs and only in the Cpy and Negate modes. Specifies the time of the fall filter, in micro-seconds. The fall filter is defined only if Fall Tme >0.
IOMsgID	UDINT	Defines the message number to associate with the actual remapping/forcing
IoClass	BYTE	Input belonging class

The VarType field can have the following values:

VarType	Mnemonic	Description
0	vtBool	The input to be remapped is bool type, that is a digital input
1	vtByte	The input to be remapped is byte type
2	vtWord	The input to be remapped is word type
3	vtDword	The input to be remapped is dword type

Concerning the InExtraRemapStructType structure, the Mode field can have the following values:

Mode	Mnemonic	Description
0	rmpIgnore	Nothing is done. Logical input is not changed.
1	rmpCopy	The physical input value is assigned to the logical input.
2	rmpSetToTrue	True value is assigned to the logical input, apart from the physical input value. This mode can be applied only with digital inputs.
3	rmpSetToFalse	False value is assigned to the logical input, apart from the physical input value. This mode can be applied only with digital inputs.
4	rmpNegate	The negative value of the physical input is assigned to the logical input, bit by bit i.e. for a byte type input, if physical input has a 16#05 hexadecimal value, the value 16#fa is assigned to the logical input
5	rmpSetWithValue	The value, defined in the Value field of the InExtraRemapStructType structure, is assigned to the logical input, apart from the physical input value.

The InRemapStructType structure contains the remapping definition of a digital input; this structure is made with the following fields:

Field	Type	Description
PhysWord	UINT	Specifies a physical input word. i.e. 113 values specifies the IW113 word
PhysBit	BYTE	Specifies a bit within a logical physical word defined in the PhysWord field. Can have values from 0 to 15.
LogicWord	UINT	Specifies a logical input word. i.e. value 128 specifies word MW128
LogicBit	BYTE	Specifies a bit within a logical input word defined in the LogicWord field. Can have values from 0 to 15.
Mode	BYTE	Specifies the remapping execution mode.

In the InRemapStructType structure, the Mode field can have the following values:

Mode	Mnemonic	Description
0	rmpIgnore	Nothing done. The logic input does not change.

1	rmpCopy	The PhysBit values of the input word PhysWord is assigned to the LogicBit of the LogicWord word.
2	rmpSetToTrue	The value “true” is assigned to the LogicBit bit of the LogicWord word, apart from the value of the PhysBit bit of the PhysWord input word.
3	rmpSetToFalse	The value “false” is assigned to the LogicBit bit of the LogicWord word, apart from the value of the PhysBit bit of the PhysWord input word.
4	rmpNegate	The negative value of the PhysiBit of the PhysWord input word is assigned to the LogicBit of the LogicWord word.

Return values

The following table lists all values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function execute without errors
0x001500AF	Out of range: the value of at least one parameter is out of range
0x001500B2	Parameters inconsistency

Example

```

Ret : dword;
RemapStruct : InExtraRemapStructType;

(* in the example below the ForceRemapInput function defines the copy of the 0
physical input bit of the IB113 byte on the 7 logic input bit of the word
MW64; it is defined a rise filter with 100 ms time; the fall filter is
undefined *)
RemapStruct.VarType := vtBool;
RemapStruct.PhysAddr := 113;
RemapStruct.PhysBit := 0;
RemapStruct.LogicAddr := 128;
RemapStruct.LogicBit := 7;
RemapStruct.Mode := rmpCopy;
RemapStruct.RiseTime := 100000;
RemapStruct.FallTime := 0;
Ret := ForceRemapInput (RemapStruct);

(* in this example the ForceRemapInput function defines the byte copy of an
IB50 physical input on the logic input byte MB62 *)
RemapStruct.VarType := vtByte;
RemapStruct.PhysAddr := 50;
RemapStruct.LogicAddr := 62;
RemapStruct.Mode := rmpCopy;
Ret := ForceRemapInput (RemapStruct);

(* the function RemapInput() has also to be called in order to make a copy *)
Ret := RemapInput ();

```

See also:

[RemapInput\(\)](#), [ReadRemapInputCFG\(\)](#)

10.12 OpenDB

The OpenDB function opens a database containing PLC data structures.

Syntax

```
ret_code := OpenDB (DbName, DbType, DbHandle) ;
```

Input parameters

DbName (STRING) name of the DB to open
DbType (INT) retentive memory or memory mapped file

Output parameter

DbHandle (DINT) handle opened database

Execution mode

Wait

Use

The function opens a database containing PLC data structures; the opening of a database is necessary to read/write the database records later. The database to delete is identified by name (defines on the *DbName* parameter) and by the database storage on a retentive memory or on a memory mapped file (*DbType* parameter).

In the *DbHandle* output parameter the function returns the handle to the opened database: this handle will be forwarded to the functions allowing the database read/write (ReadDB/WriteDB).

When database record reading/writing are done, database should be closed using the CloseDB function.

The following table lists all possible values that *DbType* parameter can have:

DbType	Mnemonic	Description
0	dbRetainMemory	The database to open is stored in the physic retentive memory
1	dbMemoryMappedFile	The database to open is stored in the memory mapped file

The database to open must exist therefore it should have been previously created using the CreateDB function.

Return values

The following table lists all possible values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0002	Database name error; usually the name is too long
0x003C0005	Integer error
0x003C0008	Error in the DbType parameter value
0x003C0009	Database cannot be opened as too it is opened already
0x003C000A	Database cannot be opened because it does not exist
0x003C000D	The memory mapped file containing the database to open cannot be opened
0x003C000E	The memory mapped file containing the database to open cannot be mapped
0x003C00016	Database cannot be opened as too many databases are opened already

Example

```
Ret      : dword;
DbHandle : dint;
...
(* open the Axes database, stored in the memory mapped file, returning to the
database the handle in DbHandle *)
Ret := OpenDB('Axes', dbMemoryMappedFile, DbHandle);
```

See also

[CreateDB](#), [CloseDB](#), [ReadDB](#), [WriteDB](#)

10.13 ReadDB

The ReadDB function reads a data base record containing PLC data structures.

Syntax

```
ret_code := ReadDB (VarName, RecordNum, DbHandle, DbType, Mode) ;
```

Input parameters

<i>VarName</i> (STRING)	PLC variable name where writing a record read by the database
<i>RecordNum</i> (INT)	database record number to read
<i>DbHandle</i> (DINT)	database handle
<i>DbType</i> (INT)	retentive memory or memory mapped file
<i>Mode</i> (INT)	defines the operating mode in case the PLC variable type does not match with the data structures type that can be stored in the database.

Execution mode

Wait

Use

The function reads a database record writing it in the PLC *VarName* PLC variable.

The database, containing the record to read, is identified by the handle -defined within the *DbHandle* parameter(OpenDB function output parameter)- and by the storage of the database in retentive memory or in a memory mapped file (*DbType* parameter).

The record number identifies the database record to be read is identified by the record II record (*RecordNum* parameter); *RecordNum* parameter can have values ranging from 1 (the first database record) up to the number of records in the database. The number of records in the database is defined when creating the database (through the CreateDB function) and can be changed using the ResizeDB function.

For a successful database record reading, these conditions are mandatory:

- ▷ The database to close must be present , therefore it should have been previously created using the CreateDB function
- ▷ The database has to be opened, therefore it should have been previously opened using the OpenDB function, returning the handle that identifies the database.
- ▷ The record to read must exist

The *VarName* variable must be structure type.

The *VarName* variable can be one of the following variables:

- ▷ **Global variable:** indicating the variable name in the *VarName* parameter allows to identify the variable name (for instance ‘GlobalVarName’)
- ▷ **Local variable defined within a program:** indicating the program instance followed by the variable name in the *VarName* parameter identifies the variable name (for instance ‘ProgramInstanceName.VarName’)
- ▷ **Local variable defined within a function block:** writing the program instance followed by the function block instance and then by the variable name in the *VarName* parameter, allows to identify the variable name (for instance ‘ProgramInstanceName.FunctionBlockInstanceName.VarName’).

The following table lists all possible values that *DbType* parameter can have:

DbType	Mnemonic	Description
0	dbRetainMemory	The database to open is stored in the physic retentive memory
1	dbMemoryMappedFile	The database to open is stored in the memory mapped file

The following table lists all possible values *Mode* parameter can have:

Mode	Description
0	If VarName type is different from the data structure types stored in the database, the VarName variable content does not change.
1	If VarName type is different from the data structure types stored in the database, within the VarName variable are written only fields having same type and name; all the other fields without same name and type, are reset.

Return values

The following table lists all possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0008	Error in the DbType parameter value
0x003C0012	Invalid database handle (DbHandle parameter)
0x003C0013	Non-existent record to read
0x003C0014	Error in the VarName parameter value: usually the parameter has an empty string
0x003C0015	The VarName variable does not exist within the PLC program
0x003C0018	The VarName variable type does not match with the data structure types stored in the database

Example

```

Ret      : dword;
DbHandle : dint;
...
(* reads the first database record identified by the DbHandle handle, on a
retentive memory, writing it in the GlbVar PLC global variable *)
Ret := ReadDB('GlbVar', 1, DbHandle, dbRetainMemory, 1);

```

See also

CreateDB, OpenDB, WriteDB

10.14 ReadRemapOutputCFG

The ReadRemapOutputCFG function uploads in memory a file of outputs remap definitions and deletes all definitions that could have been previously uploaded.

Syntax

```
ret_code := ReadRemapOutputCFG (Filename) ;
```

Input parameter

Filename (STRING) name, (with path), of the output, remap and definitions file

Execution mode

Wait

Use

The ReadRemapOutputCFG function uploads in memory a file defining the digital outputs remap. The file defining the digital outputs remap, complete with path, must be forwarded in the *Filename* parameter. This function allows to automatically remap logic digital outputs (MW variables) on physic digital output (OW variables).

The following output types can be remapped:

- *bool*: a digital logical output (bit of a MW variable) is remapped on a physical digital output (bit of a OW variable).
- *byte*: a logical output byte (a MB variable) is remapped on a physical output byte (a OB variable)
- *word*: a logical output word (a MW variable) is remapped on a physical output word (a OW variable)
- *dword*: a logical output dword (a ML variable) is remapped on a physical output dword (a OL variable)

Note that ReadRemapOutputCFG function uploads in memory only the output remapping definition file, without actually running the remap. For a real output remap, the function RemapOutput() has to be called.

Each remapping definition has a mode (*Mode*) specifying how to execute the remapping. The existing modes are:

Mode	Description
Copy	The logical output value is assigned to the physical output
Ignore	The physical output does not change
SetToTrue	The “true” value is assigned to the physical output, apart from the logical output value. This mode can be applied only to digital outputs.
SetToFalse	The “false” value is assigned to the physical output, apart from the logical output value. This mode can be applied only to digital outputs.
Negate	The negative value of the logical output is assigned to the physical output, bit by bit
SetToValue	A fixed value is assigned to the physical output, apart from the logical output value. The value to set is defined by the Value attribute and can be expresses both in decimal and hexadecimal format. i.e. “30” = “0x1E”

Note that the function **does NOT** check the consistency in case different remapping functions lead to write the same output area. For instance, if a definition leads to write the OW10 word and another definition leads to write the OB20 byte, the function does not return any error, even if the two writings conflict. The number of outputs to be remapped cannot exceed the *MaxLineNumber* field value, defined in the first lines of the XML file.

If the *ReadRemapOutputCFG* function is called more than once, the last definitions are NOT incrementally “added” to those of the previous calls. At every function call, in fact, all the remap definitions are deleted and then uploaded in memory only the definitions on the past file , as function call parameter.

The outputs remap definition file must be a XML file with a defined syntax. There are two different file syntaxes; the file version is defined by the *FileVersion* attribute:

- ▷ *FileVersion* 2. This syntax is the most recent and it can be used only starting from the OPEN 3.1 version
- ▷ *FileVersion* 1. This syntax can be used also for versions previous to the OPEN 3.1, but with some restrictions compared to the version 2; only the consistency with the past is kept. The restrictions are:
 1. Only digital outputs are managed (Boolean type); the byte, word and dword output types are not managed
 2. Not managed in SetToValue mode

This is an example of a outputs remap definition XML file, in version 2, the most recent and complete:

```
<?xml version="1.0" encoding="UTF-8" ?>
<OutputRemapConfig FileVersion="2" MaxLineNumber="128">
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="0" LogicAddr="23" LogicBit="15"
    Mode="Copy"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="1" LogicAddr="23" LogicBit="14"
    Mode="Ignore"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="2" LogicAddr="23" LogicBit="13"
    Mode="SetToTrue"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="3" LogicAddr="23" LogicBit="12"
    Mode="SetToFalse"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="4" LogicAddr="23" LogicBit="11"
    Mode="Negate"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="5" LogicAddr="23" LogicBit="10"
    Mode="SetToValue" Value="1"/>
  <OutRemap VarType="byte" PhysAddr="4" LogicAddr="23" Mode="Negate"/>
  <OutRemap VarType="byte" PhysAddr="5" LogicAddr="24" Mode="SetToValue"
    Value="0x0f"/>
  <OutRemap VarType="word" PhysAddr="6" LogicAddr="25" Mode="Copy"/>
  <OutRemap VarType="word" PhysAddr="7" LogicAddr="26" Mode="SetToValue"
    Value="7745"/>
  <OutRemap VarType="dword" PhysAddr="8" LogicAddr="27" Mode="Copy"/>
  <OutRemap VarType="dword" PhysAddr="9" LogicAddr="28" Mode="SetToValue"
    Value="0x1234567A"/>
</OutputRemapConfig>
```

Let's see some examples in detail:

The definition

```
<OutRemap VarType="bool" PhysAddr="99" PhysBit="0" LogicAddr="23" LogicBit="15"
Mode="Copy"/>
```

Automatically assignes a logical output bit (bit 15 of the word MW23) to a physical output bit (bit 0 of the word OW99).

The definition

```
<OutRemap VarType="byte" PhysAddr="4" LogicAddr="23" Mode="Negate"/>
```

Automatically assignes the negative value of the MB23 logical output byte to the OB4 physical output byte.

The definition

```
<OutRemap VarType="word" PhysAddr="6" LogicAddr="24" Mode="Copy"/>
```

Automatically assignes the value of the MW24 logical output word to the OW6 physical output word.

The definition

```
<OutRemap VarType="dword" PhysAddr="9" LogicAddr="25" Mode="SetToValue"
Value="0x1234567A"/>
```

Automatically assignes the hexadecimal value 1234567A to a OL9 physical output dword.

This is an example of an output remap definition XML file, in version 1, the oldest, kept to be consistent with the past:

```
<?xml version="1.0" encoding="UTF-8" ?>
<OutputRemapConfig FileVersion="1" MaxLineNumber="128">
  <OutRemap PhysWord="99" PhysBit="0" LogicWord="23" LogicBit="15" Mode="Copy"/>
  <OutRemap PhysWord="99" PhysBit="1" LogicWord="23" LogicBit="14" Mode="Ignore"/>
  <OutRemap PhysWord="99" PhysBit="2" LogicWord="23" LogicBit="13" Mode="SetToTrue"/>
  <OutRemap PhysWord="99" PhysBit="3" LogicWord="23" LogicBit="12" Mode="SetToFalse"/>
  <OutRemap PhysWord="99" PhysBit="4" LogicWord="23" LogicBit="11" Mode="Negate"/>
</OutputRemapConfig>
```

For instance, the line

```
<OutRemap PhysWord="88" PhysBit="0" LogicWord="24" LogicBit="15" Mode="Copy"/>
```

Allows to automatically copy a logical output bit (the bit 15 MW24 word) on a physical output bit (bit 0 of the OW88 word).

Return values

The following table lists all possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0015006A	Memory allocation error
0x001500AA	The file name in Filename is too long
0x001500AB	The file name in Filename does not exist
0x001500AC	Error in uploading the field specified in the Filename parameter
0x001500AD	System cannot read the file as the file version (defined by FileVersion attribute within the file) is too advanced
0x001500AE	Syntax error within the XML file

0x001500AF	Within the file there are fields having values exceeding the range allowed.
0x001500B0	The number of file lines (number of bits to remap) is higher than MaxLineNumber field value defined in the first lines of the XML file.
0x001500B2	Inconsistency among attributes

Example

```
Ret : dword;  
Ret := ReadRemapOutputCFG ('\\SSD\\osai\\OutputRemapFile.xml');
```

See also

[RemapOutput\(\)](#), [ForceRemapOutput\(\)](#)

10.15 AddRemapOutputCFG

The AddRemapOutputCFG function uploads in memory a file of output remap definitions, without deleting the definitions previously uploaded.

Syntax

```
ret_code := AddRemapOutputCFG (Filename) ;
```

Input parameters

Filename (STRING) name, complete with path, output remap definitions file

Execution mode

Wait

Use

The AddRemapOutputCFG function uploads in memory an output remap definitions file. The name of the output remap definitions file, with its path, has to be passed to the *Filename* parameter.

If the AddRemapOutputCFG function is called more than once, the last definitions are incrementally added to those referring to the previous calls.

This function automatically remaps logical outputs (MW variables) on physical output (OW variables). The output types below can be remapped:

- *bool*: a digital logical output (bit of a MW variable) is remapped on a physical digital output (bit of a OW variable).
- *byte*: a logical output byte (a MB variable) is remapped on a physical output byte (a OB variable)
- *word*: a logical output word (a MW variable) is remapped on a physical output word (a OW variable)
- *dword*: a logical output dword (a ML variable) is remapped on a physical output dword (a OL variable)

Note that AddRemapOutputCFG function uploads in memory only the output remapping definition file, without actually running the remap. For a real output remap, the function RemapOutput() has to be called.

Each remapping definition has a mode (*Mode*) specifying how to execute the remapping. The existing modes are:

Mode	Description
Copy	The logical output value is assigned to the physical output
Ignore	The physical output does not change
SetToTrue	The “true” value is assigned to the physical output, apart from the logical output value. This mode can be applied only to digital outputs.
SetToFalse	The “false” value is assigned to the physical output, apart from the logical output value. This mode can be applied only to digital outputs.
Negate	The negative value of the logical output is assigned to the physical output, bit by bit
SetToValue	A fixed value is assigned to the physical output, apart from the logical output value. The value to set is defined by the Value attribute and can be expressed both in decimal and hexadecimal format. i.e. “30” = “0x1E”

Note that the function does NOT check the consistency in case different remapping functions lead to write the same physical output area. For instance, if a definition leads to write the OW10 word and another definition leads to write the OB20 byte, the function does not return any error, even if the two writings conflict. The number of outputs to be remapped cannot exceed the *MaxLineNumber* field value, defined in the first lines of the XML file.

The outputs remap definition file must be a XML file with a defined syntax. There are two different file syntaxes; the file version is defined by the *FileVersion* attribute:

- ▷ *FileVersion* 2. This syntax is the most recent and it can be used only starting from the OPEN 3.1 version
- ▷ *FileVersion* 1. This syntax can be used also for versions previous to the OPENcontrol 3.1, but with some restrictions compared to the version 2; only the consistency with the past is kept. The restrictions are:
 3. Only digital outputs are managed (Boolean type); the byte, word and dword output types are not managed
 4. Not managed in SetToValue mode

This is an example of a outputs remap definition XML file, in version 2, the most recent and complete:

```
<?xml version="1.0" encoding="UTF-8" ?>
<OutputRemapConfig FileVersion="2" MaxLineNumber="128">
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="0" LogicAddr="23" LogicBit="15"
    Mode="Copy"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="1" LogicAddr="23" LogicBit="14"
    Mode="Ignore"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="2" LogicAddr="23" LogicBit="13"
    Mode="SetToTrue"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="3" LogicAddr="23" LogicBit="12"
    Mode="SetToFalse"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="4" LogicAddr="23" LogicBit="11"
    Mode="Negate"/>
  <OutRemap VarType="bool" PhysAddr="99" PhysBit="5" LogicAddr="23" LogicBit="10"
    Mode="SetToValue" Value="1"/>
  <OutRemap VarType="byte" PhysAddr="4" LogicAddr="23" Mode="Negate"/>
  <OutRemap VarType="byte" PhysAddr="5" LogicAddr="24" Mode="SetToValue"
    Value="0x0f"/>
  <OutRemap VarType="word" PhysAddr="6" LogicAddr="25" Mode="Copy"/>
  <OutRemap VarType="word" PhysAddr="7" LogicAddr="26" Mode="SetToValue"
    Value="7745"/>
  <OutRemap VarType="dword" PhysAddr="8" LogicAddr="27" Mode="Copy"/>
  <OutRemap VarType="dword" PhysAddr="9" LogicAddr="28" Mode="SetToValue"
    Value="0x1234567A"/>
</OutputRemapConfig>
```

Let's see some examples more in detail.

The definition

```
<OutRemap VarType="bool" PhysAddr="99" PhysBit="0" LogicAddr="23" LogicBit="15"
Mode="Copy"/>
```

Automatically assignes to a physical output bit (bit 0 of the word OW99) a logical output bit (bit 15 of the word MW23).

The definition

```
<OutRemap VarType="byte" PhysAddr="4" LogicAddr="23" Mode="Negate"/>
```

Automatically assignes to the OB4 physical output byte the negative value of the MB23 logical output byte.

The definition

```
<OutRemap VarType="word" PhysAddr="6" LogicAddr="24" Mode="Copy"/>
```

Automatically assignesto the OW6 physical output word the value of the MW24 logical output word.

The definition

```
<OutRemap VarType="dword" PhysAddr="9" LogicAddr="25" Mode="SetToValue"
Value="0x1234567A"/>
```

Automatically assignes to the dword of an OL9 physical output the hexadecimal value 1234567A

This is an example of a output remap definition XML file, in version 1, the oldest, kept to be consistent with the past:

```
<?xml version="1.0" encoding="UTF-8" ?>
<OutputRemapConfig FileVersion="1" MaxLineNumber="128">
  <OutRemap PhysWord="99" PhysBit="0" LogicWord="23" LogicBit="15" Mode="Copy"/>
  <OutRemap PhysWord="99" PhysBit="1" LogicWord="23" LogicBit="14" Mode="Ignore"/>
  <OutRemap PhysWord="99" PhysBit="2" LogicWord="23" LogicBit="13" Mode="SetToTrue"/>
  <OutRemap PhysWord="99" PhysBit="3" LogicWord="23" LogicBit="12" Mode="SetToFalse"/>
  <OutRemap PhysWord="99" PhysBit="4" LogicWord="23" LogicBit="11" Mode="Negate"/>
</OutputRemapConfig>
```

For instance, the line

```
<OutRemap PhysWord="88" PhysBit="0" LogicWord="24" LogicBit="15" Mode="Copy"/>
```

allows to automatically copy a logical output bit (the bit 15 MW24 word) on a physical output bit (bit 0 of the OW88 word).

Return values

The following table lists all values that the *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0015006A	Memory allocation error
0x001500AA	The file name in Filename is too long
0x001500AB	The file name in Filename does not exist
0x001500AC	Error in uploading the field specified in the Filename parameter
0x001500AD	System cannot read the file as the file version (defined by FileVersion attribute within the file) is too advanced

0x001500AE	Syntax error within the XML file
0x001500AF	Within the file there are fields having values exceeding the range allowed.
0x001500B0	The number of file lines (number of bits to remap) is higher than MaxLineNumber field value defined in the first lines of the XML file.
0x001500B2	Attributes inconsistency

Example

```
Ret : dword;  
Ret := AddRemapOutputCFG ('\\SSD\\osai\\OutputRemapFile.xml');
```

See also

[RemapOutput\(\)](#), [ForceRemapOutput\(\)](#), [ReadRemapOutputCFG\(\)](#), [DeleteRemapOutput\(\)](#)

10.16 RemapOutput

The RemapOutput function remaps the digital outputs.

Syntax

```
ret_code := RemapOutput () ;
```

Execution mode

Wait

Use

The RemapOutput function remaps the outputs of any belonging class; in particular, it automatically remaps logic output (M variables) on physical output (O variables).

The function applies the digital outputs remap definitions previously uploaded in memory.

The outputs remap definitions can be uploaded in the memory as follow:

- ▷ Using the ReadRemapOutputCFG function that reads a group of definitions from a file
- ▷ Using the AddRemapOutputCFG function, reading a group of definitions from a
- ▷ Using the ForceRemapOutput function that uploads in the memory the remap definition of a output.

For a description of the different remapping types available, refer to the documentation the ReadRemapOutputCFG, AddRemapOutputCFG and ForceRemapOutput functions.

Return values

The following table lists all possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0X001500B1	The memory does not have any definition of the digital outputs remap.

Example

```
Ret : dword;  
Ret := RemapOutput ();
```

See also

[ReadRemapOutputCFG\(\)](#), [AddRemapOutputCFG\(\)](#), [ForceRemapOutput\(\)](#), [DeleteRemapOutput\(\)](#),
[RemapOutputByClass\(\)](#)

10.17 RemapOutputByClass

The RemapOutputByClass function remaps the outputs according to their belonging class.

Syntax

```
ret_code := RemapOutputByClass (IoClass) ;
```

Input parameters

IoClass (INT) belonging class of the outputs to remap

Execution mode

Wait

Use

The RemapOutputByClass function remaps the outputs belonging to the class defined using the *IoClass* parameter; in particular, it automatically remaps the logical output (M variables) on the physical outputs (O variables).

The function applies the output remap definitions previously uploaded in memory. These definition can be uploaded in the following ways:

- ▷ Using the ReadRemapOutputCFG function, that reads a group of definitions from a file
- ▷ Using the AddRemapOutputCFG function, that reads a group of definitions from a file
- ▷ Using the ForceRemapOutput function, that upload in memory an output remap definition

The belonging class of every signal can be specified in the following functions.

Check the ReadRemapOutputCFG, AddRemapOutputCFG, and ForceRemapOutput functions documentation to get a description of all the types of remapping that can be used.

Return values

The following table lists all values that *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0X001500B1	There is no outputs remap definition in the memory

Example

```
Ret : dword;
Ret := RemapOutputByClass (1);
```

See also

ReadRemapOutputCFG(), AddRemapOutputCFG(), ForceRemapOutput(), DeleteRemapOutput(), RemapOutput()

10.18 DeleteRemapOutput

The DeleteRemapOutput function deletes from memory all the remap definitions of the outputs.

Syntax

```
ret_code := DeleteRemapOutput () ;
```

Execution mode

Wait

Use

The DeleteRemapOutput function deletes from memory all the remap definitions of the outputs.

The function deletes the output remap definitions previously uploaded in memory. These definition can be uploaded in the following ways:

- › Using the ReadRemapOutputCFG function, that reads a group of definitions from a file
- › Using the AddRemapOutputCFG function, that reads a group of definitions from a file
- › Using the ForceRemapOutput function, that upload in memory an output remap definition

Return value

The following table lists all values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example

```
Ret : dword;  
Ret := DeleteRemapOutput () ;
```

See also

[ReadRemapOutputCFG\(\)](#), [AddRemapOutputCFG\(\)](#), [ForceRemapOutput\(\)](#), [RemapOutput\(\)](#)

10.19 ResizeDB

The ResizeDB function changes the number of records in a database with PLC data structures.

Syntax

```
ret_code := ResizeDB (DbName, DbType, NewRecordsNum) ;
```

Input parameters

<i>DbName</i> (STRING)	DB name to be resized
<i>DbType</i> (INT)	Retentive memory or memory mapped file
<i>NewRecordsNum</i> (INT)	New number of records in the database

Execution mode

Wait

Use

The function allows to change the number of records in a database containing PLC data structures. The database to resize is identified by name (defined in the *DbName* parameter) and by the fact that the database is stored in retentive memory or in a (*DbType* parameter). The *NewRecordsNum* parameter indicates the new number of records within the database.

The following table lists all possible values *DbType* parameter can have:

DbType	Mnemonic	Description
0	dbRetainMemory	The database to resize is stored in retentive
1	dbMemoryMappedFile	The database to resize is stored in the memory mapped file

The database to resize has to be previously created using the CreateDB function.

Return values

The following table lists all possible values *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Retentive memory unavailable
0x003C0004	The retentive memory does not have enough space to store the database with the new number of records
0x003C0005	Internal error
0x003C0007	No records can be added in the memory mapped file containing the database to resize
0x003C0008	Error in the <i>DbType</i> parameter value
0x003C0009	Database cannot be resized as it is opened (see OpenDB function)
0x003C000A	The database cannot be resize as the database does not exist
0x003C000C	The number of records in a database cannot be reduced
0x003C000D	The memory mapped file containing the database to resize cannot be opened
0x003C000E	The memory mapped file, containing the database to resize, cannot be mapped in the memory.

Example

```
Ret      : dword;  
...  
(* resizes the database Axes, stored in the memory mapped file, increasing up  
to 20 the records number *)  
Ret := ResizeDB('Axes', dbMemoryMappedFile, 20);
```

See also

[CreateDB](#), [OpenDB](#), [RenameDB](#)

10.20 CopyDB

The CopyDB function copies a database containing all PLC data structures.

Syntax

```
ret_code := CopyDB (SourceDbName, SourceDbType, DestDbName, DestDbType, Mode) ;
```

Input parameters

<i>SourceDbName</i> (STRING)	DB name to copy
<i>SourceDbType</i> (INT)	DB type to copy (retentive memory or memory mapped file)
<i>DestDbName</i> (STRING)	name of the destination DB
<i>DestDbType</i> (INT)	type of the destination DB (retentive memory or memory mapped file)
<i>Mode</i> (INT)	sets what to do if destination database already exists

Execution mode

Wait

Use

The function allows to copy a database containing PLC data structures.

The copy can be made:

- ▷ From retentive memory to retentive memory
- ▷ From retentive memory to memory mapped file
- ▷ From memory mapped file to retentive memory
- ▷ From memory mapped file to memory mapped file

The database to copy is identified by the name, defined in the *SourceDbName* parameter, and by the fact that the database is stored in retentive memory or memory mapped file (*SourceDbType* parameter).

The destination database is identified by the name defined in *DestDbName* parameter, and by the fact that the database is stored in retentive memory or memory mapped file (*DestDbType* parameter).

The following table lists all the possible values *SourceDbType* and *DestDbType* parameters can have

DbType	Mnemonic	Description
0	dbRetainMemory	The database to resize is stored in retentive
1	dbMemoryMappedFile	The database to resize is stored in the memory mapped file

The following table lists all possible values *Mode* parameter can have:

Mode	Description
0	The function aborts if the destination database already exists
1	If the destination database already exists, the function overwrites it

The database to copy must be previously created using the CreateDB function.

Return values

The following table lists all possible values *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Retentive memory unavailable
0x003C0002	Database error name: name is too long
0x003C0003	The DestDbName database already exists
0x003C0004	The retentive memory does not have enough space to store the database with the new number of records
0x003C0005	Internal error
0x003C0006	Error in creating the memory mapped file containing the destination database
0x003C0007	Error in writing the memory mapped file containing the destination database
0x003C0008	SourceDbType parameter value errors or error in the DestDbType parameter value
0x003C0009	The database cannot be copied as the destination database is opened
0x003C000A	The database cannot be copied as the origin database does not exist
0x003C000D	The memory mapped file containing the database to resize cannot be opened
0x003C000E	The memory mapped file, containing the database to resize, cannot be mapped in the memory.
0x003C000F	Error in the memory mapped file copy containing the database

Example

```
Ret      : dword;
...
(* copies the Axes database, stored on memory mapped file, on the Axes2
database, stored on retentive memory; if Axes2 database already exists the
function aborts *)
Ret := CopyDB('Axes', dbMemoryMappedFile, 'Axes2', dbRetainMemory, 0);
```

See also

[CreateDB](#), [OpenDB](#), [RenameDB](#)

10.21 CreateDB

The CreateDB creates a database to store PLC data structures.

Syntax

```
ret_code := CreateDB (VarName, RecordsNum, DbName, DbType, Mode) ;
```

Input parameters

<i>VarName</i> (STRING)	PLC variable name
<i>RecordsNum</i> (INT)	number of records to allocate
<i>DbName</i> (STRING)	database name
<i>DbType</i> (INT)	retentive memory or memory mapped file
<i>Mode</i> (INT)	sets what to do if the destination database already exists

Input parameter

Execution mode

Wait

Use

The function Creates a database that can be used to store PLC data structures. Each database is identified by its name, defined in the *DbName* parameter.

A database contains a homogeneous group of records; the number of records in the database is defined by the database creation (*RecordsNum* parameter).

A database can be stored (according to the *DbType* parameter value) in the retentive memory or in the memory mapped file; in the second case, the database will be stored in a file named *DbName* and DB extension in the tables directory (...\\OEM\\tbl). Each database record contains the data structures associated to the PLC variable *VarName*.

The structure type of *VarName* variable must be:

- ▷ Simple structures containing scalar type data
- ▷ Complex structures containing: other structures, structures array, scalar types array, strings.

A record in the database CANNOT contain the following data structures (structures can be stored only as structure fields):

- ▷ Array
- ▷ Scalar types
- ▷ Strings

The *VarName* variable can be one of the following variables:

- ▷ A global variable: the variable can be identified indicating the variable name in the *VarName* parameter, i.e. 'GlobalVarName'
- ▷ A local variable defined within a program: the variable can be identified writing the program instance name, followed by the variable name, in the *VarName* parameter, i.e. 'ProgramInstanceName.VarName'

- ▷ A local variable defined within a function block: the variable can be identified by writing in the *VarName* parameter the program instance name followed by the function block instance name, followed by the variable name, i.e. 'ProgramInstanceName.FunctionBlockInstanceName.VarName'

The following table lists all the possible values *DbType* parameter can have

DbType	Mnemonic	Description
0	dbRetainMemory	The database will be stored in the retentive memory (if available)
1	dbMemoryMappedFile	The database is stored in the memory mapped file

The following table lists all possible values *Mode* parameter can have:

Mode	Description
0	The function aborts if the destination database already exists
1	If the destination database already exists, the function deletes and re-writes it

Return values

The following table lists all possible values *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0002	Error in the database name; usually the name is too long
0x003C0003	The database to create already exists
0x003C0004	The retentive memory does not have space enough to store the database to create
0x003C0005	Internal error
0x003C0006	Error in the memory mapped file creation
0x003C0007	Error in the memory mapped file writing
0x003C0008	Error in the DbType parameter value
0x003C0014	VarName parameter value error: usually the parameter contains the empty string
0x003C0015	The VarName does not exist within the PLC program

Example

```
Ret      : dword;
...
(* creates the Axes database on the memory mapped file; the database has 10
records. Each record data structure is the one of the global PLC variable named
GlbVar *)
Ret := CreateDB('GlbVar', 10, 'Axes', dbMemoryMappedFile, 1);
```

See also

[OpenDB](#), [WriteDB](#), [ReadDB](#), [DeleteDB](#), [RenameDB](#), [ResizeDB](#)

10.22 FileListDB

The FileListDB function provides the existing databases list.

Syntax

```
ret_code := FileListDB (DbNamesList, MaxDbNum, DbType, DbNum) ;
```

Input/output parameter

DbNamesList (ARRAY OF STRING) contains the present databases list

Input parameters

MaxDbNum (INT) maximum number of databases that can be inserted from the function in
DbNamesList
DbType (INT) database type (retentive memory or memory mapped file)

Output parameters

DbNum (INT) number of databases inserted by the function in *DbNamesList*

Execution mode

Wait

Use

The function provides the list of the database names in the retentive memory or in the file system according to the *DbType* parameter.

The function places in the *DbNamesList* array the existing database names list, placing in the *DbNum* output parameter the number of databases inserted in the array.

The following table lists all possible values *DbType* parameter can have:

DbType	Mnemonic	Description
0	dbRetainMemory	The function provides the database names list existing in the retentive memory.
1	dbMemoryMappedFile	The function provides the database names list existing in the file system (memory mapped file)

Return values

The following table lists all possible values *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0008	Error in the DbType parameter value
0x003C0017	Error in the memory mapped file search in the file system

Example

```
Ret          : dword;
DbNamesList : array[1..10] of string;
DbNum      : int;
...
(* the function places (up to 10db) in DbNamesList the list of the existing
databases names in the retentive memory; the function places in DbNum the
number of databases inserted in DbNamesList *)
Ret := FileListDB(DbNamesList, 10, dbRetainMemory, DbNum);
```

See also

CreateDB, OpenDB

10.23 ForceRemapOutput

The ForceRemapOutput function allows to define the remapping of logical digital output on a physical digital output.

Syntax

```
ret_code := ForceRemapOutput (RemapStruct) ;
```

Input parameters

RemapStruct (ANY) structure defining a digital output remap

Possible overloads

ForceRemapOutput (OutRemapStructType)

ForceRemapOutput (OutExtraRemapStructType)

Execution mode

Wait

Use

The ForceRemapOutput function allows to define the remapping of a logical output (a M variable) on an physical output (O variable). The output types below can be remapped:

- ▷ *bool*: a digital logical output is remapped on a physical digital output
- ▷ *byte*: a logical output byte is remapped on a physical output byte
- ▷ *word*: a logical output word is remapped on a physical output word
- ▷ *dword*: a logical output dword is remapped on a physical output dword

The function uploads in memory the remapping output definition:

- ▷ if there is already in memory a remapping definition referred to the same physical output (because of previous calls to the ReadRemapOutputCFG or to the ForceRemapOutput function), then the ForceRemapOutput overwrites the defintion.
- ▷ If there is no remapping definition in memory referred to the same physical output, then the definition is added to others already existing.

Note that ForceRemapOutput function uploads in memory only the output remapping definition file, without actually running the remap. For a real output remap, the function RemapOutput() has to be called.

Each element can be associated to a message identifier (*IOMsgID* - to be used for viewing a message within the viewing tool of the remappings/forcings) and to an *IOClass*, used to execute the RemapOutput function only for the Outputs belonging to a specific class.

Note that the function does NOT check the consistency in case different remapping functions lead to write the same physical output area. For instance, if a definition leads to write the OW10 word and another definition leads to write the OB20 byte, the function does not return any error, even if the two writings conflict. The number of outputs to be remapped cannot exceed the *MaxLineNumber* field value, defined in the first lines of the XML file.

Each remapping definition has a mode (*Mode*) specifying how to execute the remapping. The existing modes are:

- Ignore: the physical output does not change
- Copy: the logical output value is assigned to the physical output
- SetToTrue: the “true” value is assigned to the physical output, apart from the logical output value. This mode can be applied only to digital outputs.
- SetToFalse: the “false” value is assigned to the physical output, apart from the logical output value. This mode can be applied only to digital outputs.
- Negate: the negative value of the logical output is assigned to the physical output, bit by bit
- SetToValue: a fixed value is assigned to the physical output, apart from the logical output value.

The remapping definition can be made passing one of the structures below to the function:

- *OutExtraRemapStructType*. This structure can be used only starting from OPENcontrol 3.1.
- *OutRemapStructType*. This structure can be used also for version previous to OPENcontrol 3.1, but with some restrictions in using the structure *OutExtraRemapStructType*, and it is kept only the consistency with the past. The restrictions are:
 1. Only digital outputs are managed (Boolean type); the byte, word and dword output types are not managed
 2. Not managed in SetToValue mode

The *OutExtraRemapStructType* structure contains the definition of the output remapping; this structure is made by following fields:

Field	Type	Description
VarType	BYTE	Defines the input type to be remapped: bool, byte, word, dword
PhysAddr	UINT	Specifies in bytes the offset in the physical inputs area. i.e. for a byte type physical input the value 13 specifies the IB113 byte
PhysBit	BYTE	For a bool type input, it specifies a bit within the physical input byte defined in the PhysAddr field. Can have value from 0 to 7.
LogicAddr	UINT	Specifies the offset in byte in the logical inputs area. i.e. for a word type logical input, the value 128 specifies the MW64 word.
LogicBit	BYTE	For a bool type input, it specifies a bit in the logical input byte defined in the LogicAddr field. Can have values form 0 to 7.
Mode	BYTE	Specifies the remapping execution mode
Value	UDINT	It is important only in SetToValue mode. It specifies the value assigned to the logical input.
IOMsgID	UDINT	Defines the message number to associate with the actual remapping/forcing
IoClass	BYTE	Input belonging class

The VarType field can have the following values:

VarType	Mnemonic	Description
0	vtBool	The output to be remapped is bool type, that is a digital input
1	vtByte	The output to be remapped is byte type
2	vtWord	The output to be remapped is word type
3	vtDword	The output to be remapped is dword type

Concerning the *OutExtraRemapStructType* structure, the Mode field can have the following values:

Mode	Mnemonic	Description
0	rmpIgnore	Nothing is done. Logical output is not changed.
1	rmpCopy	The physical output value is assigned to the logical output.
2	rmpSetToTrue	True value is assigned to the logical output, apart from the physical output value. This mode can be applied only with digital output.
3	rmpSetToFalse	False value is assigned to the logical output, apart from the physical output value. This mode can be applied only with digital outputs.
4	rmpNegate	The negative value of the physical output is assigned to the logical output, bit by bit i.e. for a byte type output, if physical output has a 16#1F05 hexadecimal value, the value 16#E0FA is assigned to the logical output
5	rmpSetValue	The value, defined in the Value field of the <i>OutExtraRemapStructType</i> structure, is assigned to the logical output, apart from the physical output value.

The *OutRemapStructType* structure contains the remapping definition of a digital output; this structure is made by the fields below:

FIELD	Type	Description
PhysWord	UINT	Specifies a physical output word. i.e. the value 105 specifies the OW105 word.
PhysBit	BYTE	Specifies a bit within a physical output word defined in the PhysWord field. Can have values from 0 to 15.
LogicWord	UINT	Specifies an output logical word. i.e.. the value 128 specifies the word MW128
LogicBit	BYTE	Specifies a bit within a logic output word defined in the LogicWord field. Can have values from 0 to 15.
Mode	BYTE	Specifies the remapping execution mode

The field Mode can have the following values:

Mode	Mnemonic	Description
0	rmpIgnore	Nothing done The physical output does not change.
1	rmpCopy	The LogicBit of the LogicWord word value is assigned to the PhysBit of the PhysWord output word.
2	rmpSetToTrue	The value “true” is assigned to the PhysBit bit of the output PhysWord word, apart from the LogicBit bit value of the LogicWord word.
3	rmpSetToFalse	The value “false” is assigned to the PhysBit bit of the output PhysWord word, apart forform the LogicBit bit value of the LogicWord word.
4	rmpNegate	The negative LogicBit bit value of the LogicWord word is assigned to the PhysBit bit of the output PhysWord word.

Return values

The following table lists all values *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x001500AF	Out of range: at least one of the parameters has a value out of the range allowed
0x001500B2	Parameters inconsistency

Example

```

Ret : dword;
RemapStruct : OutExtraRemapStructType;

(* in the example below the ForceRemapOutput function defines the copy of the 7
logical output bit of the word MW64 on the physical output bit 2 of the byte
OB113 *)
RemapStruct.VarType := vtBool;
RemapStruct.PhysAddr := 113;
RemapStruct.PhysBit := 2;
RemapStruct.LogicAddr := 128;
RemapStruct.LogicBit := 7;
RemapStruct.Mode := rmpCopy;
Ret := ForceRemapOutput (RemapStruct);

(* in the example below the ForceRemapOutput function defines the setting of
the physical output dword OL12 at value 357 *)
RemapStruct.VarType := vtDword;
RemapStruct.PhysAddr := 48;
RemapStruct.Value := 357;
RemapStruct.Mode := rmpSetToValue;
Ret := ForceRemapOutput (RemapStruct);

(* the function RemapOutput(): has to be called to copy*)
Ret := RemapOutput ();

```

See also

[RemapOutput\(\)](#), [ReadRemapOutputCFG\(\)](#)

10.24 PurgeRetainDB

The PurgeRetainDB function purges the retentive memory recovering unused spaces, if any.

Syntax

```
ret_code := PurgeRetainDB () ;
```

Execution mode

Wait

Use

The PurgeRetainDB function rearranges and repacks the retentive memory to recover unused spaces that can be created after the database delete. When function is called, no databases must be opened with the OpenDB function in the retentive memory.

The function only works on databases stored in the retentive memory, but it doesn't have any consequences on databases stored in the memory mapped file.

Return values

The following table lists all possible values that *ret_code* variable can have:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0009	The retentive memory cannot be rearranged as at least one database is opened (see OpenDB function)

Example

```
Ret      : dword;  
...  
Ret := PurgeRetainDB();
```

See also

CreateDB, OpenDB

10.25 ReadRemapInputCFG

The ReadRemapInputCFG function allows to upload in the memory remap definitions file of inputs remap and deletes all the definitions that could have been previously updated.

Syntax

```
ret_code := ReadRemapInputCFG (Filename) ;
```

Input parameters

Filename (STRING) name, complete with path, of the remap definitions of digital inputs remap

Execution mode

Wait

Use

The ReadRemapInputCFG function allows to upload in memory a definition file of input remap. The input remap definitions file name, complete with path, has to be passed to the function in the *Filename* parameter.

The ReadRemapInputCFG function allows to automatically remap physical inputs (I variables) on logical inputs (M variables). The output types below can be remapped:

- ▷ *bool*: a digital physical input is remapped on a logical digital input
- ▷ *byte*: a physical input byte is remapped on a logical input byte
- ▷ *word*: a physical input word is remapped on a logical input word
- ▷ *dword*: a physical input dword is remapped on a logical input dword

Note that the ReadRemapInputCFG function uploads in memory only the input remapping definition, without actually running the remap. For a real remapping execution, the function RemapInput() has to be called.

Each remapping definition has a mode (*Mode*) specifying how to execute the remapping. The existing modes are:

Mode	Description
Copy	the same physical input value is assigned to the logical input
Ignore	the logical input does not change
SetToTrue	true value is assigned to the logical input, apart from the physical input value. This mode can be applied only to digital inputs.
SetToFalse	false value is assigned to the logical input, apart from the physical input value. This mode can be applied on the digital inputs.
Negate	the negative value of the physical input is assigned to the logical input, bit by bit. i.e. for a byte type input, the negative 4E is = B1
SetToValue	a fixed value is assigned to the logical input, apart from the physical input value. The value to set is defined by the Value attribute and can be expressed in decimal or hexadecimal format. i.e. “15” = “0xF”

Two filters can be defined on the digital input:

- ▷ The rise filter is characterized by *RiseTime* parameter expressed in micro-seconds. The rise filter is defined only if *RiseTime* > 0.
- ▷ The fall filter is characterized by the *FallTime* parameter, expressed in micro-seconds. The fall filter is defined only if *FallTime* > 0.

These filters can be used only in Negate and Copy modes.

In the Copy mode the filters behave as follow:

- ▷ if the rise filter is defined, after the physical input shift from false to true, the logical input shifts at true only if the physical input remains at true for a period equal -at least- to *RiseTime*
- ▷ if the fall filter is defined, after the physical input shift from true to false, the logical input shifts at false only if the physical input remains at false for a period equal -at least- to *FallTime*

In the Negate mode the filters behave as follow:

- ▷ if the rise filter is defined, after the physical input shift from true to false, the logical input shifts at true only if the physical input remains at false for a period equal -at least- to *RiseTime*
- ▷ if the fall filter is defined, after the physical input shift from false to true, the logical input shifts at false only if the physical input remains at true for a period equal -at least- to *FallTime*

Each element can be associated to a message identifier *IOMsgID* (used to view a message in the tool of the remapping/forcing view) and to an *IOClass* used to execute the *RemapInput* function only for the inputs belonging to a given class. Filters cannot be defined on other inputs as byte, word, dword.

Note that the function DOES NOT check if the different remapping definitions could write the same logical input area; for instance, if a definition writes the MW10 word and another definition writes the MB20 byte, the function does not display any error, even if the two writings conflict.

The number of inputs to be remapped cannot exceed the *MaxLineNumber* field value, defined in the first lines of the XML file. If the *ReadRemapOutputCFG* function is called more than once, the last definitions are NOT incrementally “added” to those of the previous calls. At every function call, in fact, all the remap definitions are deleted and then uploaded in memory only the definitions on the past file , as function call parameter.

The inputs remap definition file must be a XML file with a defined syntax. There are two different file syntaxes; the file version is defined by the *FileVersion* attribute:

- ▷ *FileVersion* 2. This is the most recent syntax and it can be used only starting from OPENcontrol rel. 3.1. In this case, *PhysAddr* and *LogicAddr* values represent the offsets (in byte) in the digital Input (I) and in the logical inputs (M) zones. The *PhysBit* and *LogicBit* values can vary between 0 and 7 as *bool* is considered within a byte
- ▷ *Fileversion* 1. This syntax can be used also for OPEN version previous to 3.1, but with some limitations compared to version 2, and it is kept only to comply with previous versions. The limitations are:
 1. Only digital inputs are managed (Boolean type); byte, word, dword input types are not managed
 2. Filters not manged on digital
 3. SetToValue mode not managed
 4. The *PhysWord* and *LogicWord* values represent the words in the digital input zone (IW) and in the logical inputs zone (MW). The *PhysBit* and *LogicBit* values can vary between 0 and 15 as the signals are within a word.

This is an example of XML rempar definition file of the inputs, in version 2 (the most recent and complete):

```
<?xml version="1.0" encoding="UTF-8" ?>
<InputRemapConfig FileVersion="2" MaxLineNumber="128">
    <InRemap VarType="bool" PhysAddr="99" PhysBit="0" LogicAddr="23" LogicBit="5" Mode="Copy"
        RiseTime="50000" FallTime="20000" IOMsgID ="100" IOClass = "0" />
    <InRemap VarType="bool" PhysAddr="99" PhysBit="1" LogicAddr="23" LogicBit="4" Mode="Ignore"
        IOMsgID ="101" IOClass = "0" />
    <InRemap VarType="bool" PhysAddr="99" PhysBit="2" LogicAddr="23" LogicBit="3"
        Mode="SetToTrue" IOMsgID ="102" IOClass = "0" />
    <InRemap VarType="bool" PhysAddr="99" PhysBit="3" LogicAddr="23" LogicBit="2"
        Mode="SetToFalse"/>
    <InRemap VarType="bool" PhysAddr="99" PhysBit="4" LogicAddr="23" LogicBit="1" Mode="Negate"
        RiseTime="50000" />
    <InRemap VarType="bool" PhysAddr="99" PhysBit="5" LogicAddr="23" LogicBit="0"
        Mode="SetToValue" Value="1" />
    <InRemap VarType="byte" PhysAddr="4" LogicAddr="30" Mode="Negate" />
    <InRemap VarType="byte" PhysAddr="5" LogicAddr="31" Mode="SetToValue" Value="0x0f" />
    <InRemap VarType="word" PhysAddr="6" LogicAddr="32" Mode="Copy" />
    <InRemap VarType="word" PhysAddr="7" LogicAddr="33" Mode="SetToValue" Value="7745" />
    <InRemap VarType="dword" PhysAddr="8" LogicAddr="34" Mode="Copy" IOMsgID ="105" IOClass =
        "1" />
    <InRemap VarType="dword" PhysAddr="9" LogicAddr="35" Mode="SetToValue" Value="0x1234567A"
        IOMsgID ="108" IOClass = "1" />
</InputRemapConfig>
```

Let's see some examples in detail.

The definition

```
<InRemap VarType="bool" PhysAddr="99" PhysBit="0" LogicAddr="23" LogicBit="5" Mode="Copy"
    RiseTime="50000" FallTime="20000" />
```

Automatically assignesa physical input bit (bit 5 of the byte with offset 99 of the Input area) to a logical input bit (bit 5 of the byte with offset 23 of the M variables), with a 50ms rise filter and with a 20ms fall filter.

The definition

```
<InRemap VarType="byte" PhysAddr="4" LogicAddr="33" Mode="Negate" />
```

Automatically assignesthe negative value of the IB4 physical input byte to the MB33 logical input byte.

The definition

```
<InRemap VarType="word" PhysAddr="6" LogicAddr="34" Mode="Copy" />
```

Automatically assignesthe IW3 physical input word value (offset 6) to the MW17 logical input word (offset 34).

The definition

```
<InRemap VarType="dword" PhysAddr="9" LogicAddr="36" Mode="SetToValue"
    Value="0x1234567A" />
```

Allows to automatically assigne the dword logical input ML9 (offset 36) the 1234567A hexadecimal value

This is an example of an output remap definition XML file, in version 1, the oldest, kept to be consistent with the past:

```
<?xml version="1.0" encoding="UTF-8" ?>
<InputRemapConfig FileVersion="1" MaxLineNumber="128">
    <InRemap PhysWord="99" PhysBit="0" LogicWord="23" LogicBit="15" Mode="Copy"/>
    <InRemap PhysWord="99" PhysBit="1" LogicWord="23" LogicBit="14" Mode="Ignore"/>
    <InRemap PhysWord="99" PhysBit="2" LogicWord="23" LogicBit="13" Mode="SetToTrue"/>
    <InRemap PhysWord="99" PhysBit="3" LogicWord="23" LogicBit="12" Mode="SetToFalse"/>
    <InRemap PhysWord="99" PhysBit="4" LogicWord="23" LogicBit="11" Mode="Negate"/>
</InputRemapConfig>
```

For instance, the line

```
<InRemap PhysWord="99" PhysBit="0" LogicWord="23" LogicBit="15" Mode="Copy"/>
```

Allows to automatically copy a physical input bit (bit 0 of IW99 word) on a logical input bit (bit 15 of the MW23).

Return values

The following table lists all values that *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0015006A	Error in memory allocation
0x001500AA	The name of the file in the Filename parameter is too long
0x001500AB	The file in the Filename parameter does not exist
0x001500AC	Errore nel caricamento del file specificato nel parametro Filename
0x001500AD	The system cannot read the file as the file version (defined in the FileVision attribute in the file) is too advanced
0x001500AE	Syntax error within the XML file
0x001500AF	In the file there are fields with values exceeding the range allowed
0x001500B0	The lines number of the file (number of inputs to remap) is higher than the MaxLineNumber value, defined in the first lines of the XML file.
0x001500B2	Attributes inconsistency

Example

```
Ret : dword;
Ret := ReadRemapInputCFG ('\\SSD\\osai\\InputRemapFile.xml');
```

See also

[RemapInput\(\)](#), [ForceRemapInput\(\)](#)

10.26 AddRemapInputCFG

The `AddRemapInputCFG` function allows to upload in memory a file of inputs remap definitions, without deleting the definition already uploaded.

Syntax

```
ret_code := AddRemapInputCFG (Filename) ;
```

Input parameters

Filename (STRING) name, complete with path, of the input remaps definitions file

Execution mode

Wait

Use

The `AddRemapInputCFG` function allows to upload in memory n input remap definitions file. The name of the input remap definitions file, complete with path, has to be passed to the function in the *Filename* parameter.

If the `AddRemapInputCFG` function is called more than once, the last definitions are added to those incrementally referring to the previous calls.

This function allows automatically to remap physical inputs (I variables) on logical inputs (M variables).

The input types below can be remapped:

- *bool*: a digital physical input is remapped on a logical digital input
- *byte*: a physical input byte is remapped on a logical input byte
- *word*: a physical input word is remapped on a logical input word
- *dword*: a physical input dword is remapped on a logical input dword

Note that the `ReadRemapInputCFG` function uploads in memory only the input remapping definition, without actually running the remap. For a real remapping execution, the function `RemapInput()` has to be called.

Each remapping definition has a mode (*Mode*) specifying how to execute the remapping. The existing modes are:

Mode	Description
Copy	the same physical input value is assigned to the logical input
Ignore	the logical input does not change
SetToTrue	true value is assigned to the logical input, apart from the physical input value. This mode can be applied only to digital inputs.
SetToFalse	false value is assigned to the logical input, apart from the physical input value. This mode can be applied on the digital inputs.
Negate	the negative value of the physical input is assigned to the logical input, bit by bit. i.e. for a byte type input, the negative 4E is = B1
SetToValue	a fixed value is assigned to the logical input, apart from the physical input value. The value to set is defined by the Value attribute and can be expressed in decimal or hexadecimal format. i.e. "15" = "0xF"

Two filters can be defined on each digital input:

- ▷ a rise filter characterized by the *RiseTime* attribute (expressed in micro-seconds). The rise filter is defined only if the *RiseTime* attribute is present in the XML file and is higher than zero.
- ▷ a fall filter characterized by the *FallTime* attribute (expressed in micro-seconds). The fall filter is defined only if the *FallTime* attribute is present in the XML file and is higher than zero.

These two filters can be used only in the Copy and Negate modes.

In the Copy mode the filters behave as follow:

- ▷ if the rise filter is defined, after the physical input shift from false to true, the logical input shifts at true only if the physical input remains at true for a period equal -at least- to *RiseTime*
- ▷ if the fall filter is defined, after the physical input shift from true to false, the logical input shifts at false only if the physical input remains at false for a period equal -at least- to *FallTime*

In the Negate mode the filters behave as follow:

- ▷ if the rise filter is defined, after the physical input shift from true to false, the logical input shifts at true only if the physical input remains at false for a period equal -at least- to *RiseTime*
- ▷ if the fall filter is defined, after the physical input shift from false to true, the logical input shifts at false only if the physical input remains at true for a period equal -at least- to *FallTime*

Each element can be associated to a message identifier *IOMsgID* (used to view a message in the tool of the remapping/forcing view) and to an *IOClass* used to execute the *RemapInput* function only for the inputs belonging to a given class. Filters cannot be defined on the byte, word, dword inputs type.

Note that the function does NOT check the consistency in case different remapping functions lead to write the same logical input area. For instance, if a definition leads to write the MW10 word and another definition leads to write the MB20 byte, the function does not return any error, even if the two writings conflict.

The number of inputs to be remapped cannot exceed the *MaxLineNumber* field value, defined in the first lines of the XML file.

If the *ReadRemapOutputCFG* function is called more than once, the last definitions are NOT incrementally “added” to those of the previous calls. At every function call, in fact, all the remap definitions are deleted and then uploaded in memory only the definitions on the past file , as function call parameter.

The inputs remap definition file must be a XML file with a specific syntax. For historical reasons there are two different versions of the file; the file version is defined by the *FileVersion* attribute:

- ▷ *FileVersion* 2. This syntax is the most recent and can be used only from OPENcontrol 3.1. In this case, the values *PhysAddr* and *LogicAddr* represent the offsets (in byte) namely in the digital inputs area (I) and in the logical input (M). The *PhysBit* and *LogicBit* values, can vary between 0 and 7 as *bool* is considered within a byte.
- ▷ *Fileversion* 1. This syntax can be used also for OPEN versions prior to the 3.1 release, but with some restrictions in the version 2 syntax, and it is kept only for consistency with the past. Restrictions are:
 1. Only digital inputs (Boolean type) are managed; byte, word and dword inputs type are not managed
 2. Filters are not managed on digital inputs
 3. The *SetToValue* mode is not managed
 4. The *PhysWord* and *LogicWord* values represent the words in the digital inputs area (IW) and in the logical inputs (MW). The *PhysBit* and *LogicBit* values can vary between 0 and 15 as the signals are within a word.

This is an example of a input remap definition file XML, in version 2, the most recent and complete:

```
<?xml version="1.0" encoding="UTF-8" ?>
<InputRemapConfig FileVersion="2" MaxLineNumber="128">
    <InRemap VarType="bool" PhysAddr="99" PhysBit="0" LogicAddr="23" LogicBit="5" Mode="Copy"
        RiseTime="50000" FallTime="20000" IOMsgID ="100" IOClass = "0" />
    <InRemap VarType="bool" PhysAddr="99" PhysBit="1" LogicAddr="23" LogicBit="4" Mode="Ignore"
        IOMsgID ="101" IOClass = "0" />
    <InRemap VarType="bool" PhysAddr="99" PhysBit="2" LogicAddr="23" LogicBit="3"
        Mode="SetToTrue" IOMsgID ="102" IOClass = "0" />
    <InRemap VarType="bool" PhysAddr="99" PhysBit="3" LogicAddr="23" LogicBit="2"
        Mode="SetToFalse"/>
    <InRemap VarType="bool" PhysAddr="99" PhysBit="4" LogicAddr="23" LogicBit="1" Mode="Negate"
        RiseTime="50000"/>
    <InRemap VarType="bool" PhysAddr="99" PhysBit="5" LogicAddr="23" LogicBit="0"
        Mode="SetToValue" Value="1"/>
    <InRemap VarType="byte" PhysAddr="4" LogicAddr="30" Mode="Negate"/>
    <InRemap VarType="byte" PhysAddr="5" LogicAddr="31" Mode="SetToValue" Value="0x0f"/>
    <InRemap VarType="word" PhysAddr="6" LogicAddr="32" Mode="Copy"/>
    <InRemap VarType="word" PhysAddr="7" LogicAddr="33" Mode="SetToValue" Value="7745"/>
    <InRemap VarType="dword" PhysAddr="8" LogicAddr="34" Mode="Copy" IOMsgID ="105" IOClass =
        "1" />
    <InRemap VarType="dword" PhysAddr="9" LogicAddr="35" Mode="SetToValue" Value="0x1234567A"
        IOMsgID ="108" IOClass = "1" />
</InputRemapConfig>
```

Let's see some examples in detail.

The definition

```
<InRemap VarType="bool" PhysAddr="99" PhysBit="0" LogicAddr="23" LogicBit="5" Mode="Copy"
    RiseTime="50000" FallTime="20000"/>
```

Automatically assignesa logical input bit (bit 5 of the byte at offset 23 of M variables), to a physical input bit (bit 5 of the byte at offset 99 of the Input area), with a 50ms rise filter and with a 20ms fall filter.

The definition

```
<InRemap VarType="byte" PhysAddr="4" LogicAddr="33" Mode="Negate"/>
```

Automatically assignesthe negative value of the physical IB4 input to the MB33 logical input byte.

The definition

```
<InRemap VarType="word" PhysAddr="6" LogicAddr="34" Mode="Copy"/>
```

Automatically assignesthe value of the IW3 physical input (offset 6) to the MW17 (offset 34) logical input type.

The definition

```
<InRemap VarType="dword" PhysAddr="9" LogicAddr="36" Mode="SetToValue"
    Value="0x1234567A"/>
```

Automatically assignesthe hexadecimal value 1234567A to the logical input dword ML9(offset 36).

This is an example of input remap definition XML file, in version 1 - the oldest- kept for consistency with the past:

```
<?xml version="1.0" encoding="UTF-8" ?>
<InputRemapConfig FileVersion="1" MaxLineNumber="128">
    <InRemap PhysWord="99" PhysBit="0" LogicWord="23" LogicBit="15" Mode="Copy"/>
    <InRemap PhysWord="99" PhysBit="1" LogicWord="23" LogicBit="14" Mode="Ignore"/>
    <InRemap PhysWord="99" PhysBit="2" LogicWord="23" LogicBit="13" Mode="SetToTrue"/>
    <InRemap PhysWord="99" PhysBit="3" LogicWord="23" LogicBit="12" Mode="SetToFalse"/>
    <InRemap PhysWord="99" PhysBit="4" LogicWord="23" LogicBit="11" Mode="Negate"/>
</InputRemapConfig>
```

For instance the line

```
<InRemap PhysWord="99" PhysBit="0" LogicWord="23" LogicBit="15" Mode="Copy"/>
```

Allows to automatically copy a physical input bit (bit 0 of the word IW99) on al logical input bit (bit 15 of the word MW23).

Return values

The following table lists values *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x0015006A	Memory allocation error
0x001500AA	The file name in Filename is too long
0x001500AB	The file name in Filename does not exist
0x001500AC	Error in uploading the field specified in the Filename parameter
0x001500AD	System cannot read the file as the file version (defined by FileVersion attribute within the file) is too advanced
0x001500AE	Syntax error within the XML file
0x001500AF	Within the file there are fields having values exceeding the range allowed.
0x001500B0	The number of file lines (number of bits to remap) is higher than MaxLineNumber field value defined in the first lines of the XML file.
0x001500B2	Attributes inconsistency

Example

```
Ret : dword;
Ret := AddRemapInputCFG ('\\SSD\\osai\\InputRemapFile.xml');
```

See also

[RemapInput\(\)](#), [ForceRemapInput\(\)](#), [ReadRemapInputCFG\(\)](#), [DeleteRemapInput\(\)](#)

10.27 RemapInput

The RemapInput function runs the digital input remapping.

Syntax

```
ret_code := RemapInput () ;
```

Execution mode

Wait

Use

The RemapInput function runs the digital input remapping without considering their belonging class; in particular, it automatically remaps physical inputs (I variables) on logical inputs (M variables).

The function applies the remap definitions of the input previously uploaded in memory. The inputs remap definitions can be uploaded in memory as follow:

- ▷ using the ReadRemapInputCFG function that reads a group of file definitions
- ▷ Using the AddRemapInputCFG function, that reads an group of definitions form a file
- ▷ using the ForceRemapInput function that uploads in memory a remap definition of a input.

For a description of the different remapping types, check the documentation about the ReadRemapInputCFG, AddRemapInputCFG and ForceRemapInput functions.

Return values

The following table lists all possible values *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0X001500B1	The memory does not have any remap definition of the digital outputs

Example

```
Ret : dword;
Ret := RemapInput () ;
```

See also

[ReadRemapInputCFG\(\)](#), [AddRemapInputCFG\(\)](#), [ForceRemapInput\(\)](#), [DeleteRemapInput\(\)](#), [RemapInputByClass\(\)](#)

10.28 RemapInputByClass

The RemapInput function runs the inputs remapping according to their belonging class.

Syntax

```
ret_code := := RemapInputByClass (IoClass) ;
```

Input parameter

IoClass (INT) belonging class of the inputs to remap

Execution mode

Wait

Use

The RemapInput function executes the inputs remapping apart from their belonging class; in particular, it automatically remaps physical inputs (I variables) on logical inputs (M variables).

The function applies the input remap definitions previously uploaded in memory. The input remap definitions can be uploaded in memory as follow:

- ▷ using the ReadRemapInputCFG function, that reads a group of definitions from a file
- ▷ using the AddRemapInputCFG function, that reads a group of definitions from a file
- ▷ using the ForceRemapInput function, that uploads in memory an input remap definition

Within these function, the belonging class of each signal can be specified.

For a description of the different remapping types to be used, see the documentation about the ReadRemapInputCFG, AddRemapInputCFG and ForceRemapInput functions.

Return values

The following table lists all values *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0X001500B1	No inputs remap definition in memory

Example

```
Ret : dword;
Ret := RemapInput (2);
```

See also

[ReadRemapInputCFG\(\)](#), [AddRemapInputCFG\(\)](#), [ForceRemapInput\(\)](#), [DeleteRemapInput\(\)](#),
[RemapInputByClass\(\)](#)

10.29 DeleteRemapInput

The DeleteRemapInput function deletes from memory all the inputs remap definitions.

Syntax

```
ret_code := DeleteRemapInput () ;
```

Execution mode

Wait

Use

The DeleteRemapInput function deletes from memory all the inputs remap definitions.

The function deletes the input remap definitions previously uploaded in memory. The input remap definitions can be uploaded in memory as follow:

- ▷ using the ReadRemapInputCFG function, that reads a group of definitions from a file
- ▷ using the AddRemapInputCFG function, that reads a group of definitions from a file
- ▷ using the ForceRemapInput function, that uploads in memory an input remap definition

Return value

The following table lists all values *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors

Example

```
Ret : dword;  
Ret := DeleteRemapInput () ;
```

See also

[ReadRemapInputCFG\(\)](#), [AddRemapInputCFG\(\)](#), [ForceRemapInput\(\)](#), [RemapInput\(\)](#)

10.30 GetRemapAddress

The GetRemapAddress function determines the remapping address of a physical or logical input/output.

Syntax

```
ret_code := GetRemapAddress (RemapVar, RemapAddr, RemapBit, RemapType) ;
```

Input parameter

RemapVar (ANY) Variable to which ask the remapping address

Output parameters

RemapAddr (UINT) Remapping address
RemapBit (INT) Bit where the variable is set
RemapType (INT) Variable type

Possible overloads

GetRemapAddress (BOOL, UINT, INT, INT)
GetRemapAddress (BYTE, UINT, INT, INT)
GetRemapAddress (WORD, UINT, INT, INT)
GetRemapAddress (DWORD, UINT, INT, INT)

Execution mode

Wait

Use

The GetRemapAddress function determines the remapping address of a physical or logical input/output. The address can be used in the different remapping functions. In output *RemapType* value will be given (if it is a logical input/output variable, the value is 0, 1 in case of a physical input or 2 in case of physical output).

The *RemapType* field can have the following values

RemapType	Mnemonic	Description
0	vtLogic	Input/Output Logical variable
1	vtInput	Input physical variable
2	vtOutput	Output physical variable

Return values

The following table lists all values the *ret_code* variable can have.

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0X001500B2	Inconsistency among parameters

Example

```
Ret : dword;
RemapAddr : INT;
RemapBit : INT;
RemapType : INT;
Ret := GetRemapAddr (I10_0, RemapAddr, RemapBit, RemapType);
```

See also

[RemapInput\(\)](#), [ReadRemapInputCFG\(\)](#), [RemapOutput\(\)](#), [ReadRemapOutputCFG\(\)](#)

10.31 RenameDB

The RenameDB function renames a database containing PLC data structures.

Syntax

```
ret_code := RenameDB (CurrDbName, NewDbName, DbType) ;
```

Input parameters

<i>CurrDbName</i> (STRING)	current name of the DB to rename
<i>NewDbName</i> (STRING)	new DB name
<i>DbType</i> (INT)	retentive memory or memory mapped file

Execution mode

Wait

Use

The function allows to rename a database containing PLC data structures. The database to rename is identified by the name (defined by *DbType*) and by the fact that the database is stored in a retentive memory or in the memory mapped file (*DbType* parameter). The *NewDbName* parameter specifies the new name the function will give to the database.

The following table lists all possible values *DbType* parameter can have:

DbType	Mnemonic	Description
0	dbRetainMemory	The database to rename is stored in the retentive memory
1	dbMemoryMappedFile	The database to rename is stored in the memory mapped file

The database to rename must be previously created using the CreateDB function.

Return values

The following table lists all possible values *ret_code* variable can have.

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0002	Error in the database name; usually the name is too long
0x003C0003	The NewDbName database already exists
0x003C0005	Internal error
0x003C0006	Error in creating the memory mapped file
0x003C0008	Error in the DbType parameter value
0x003C0009	Database cannot be renamed as it is opened (see OpenDB function)
0x003C000A	Database cannot be renamed as it does not exist
0x003C000B	The memory mapped file containing the database cannot be renamed

Example

```
Ret      : dword;  
...  
(* renames the Axes (stored in memory mapped file) giving it the name Axes2 *)  
Ret := RenameDB('Axes', 'Axes2', dbMemoryMappedFile);
```

See also

CreateDB, OpenDB, ResizeDB

10.32 WriteDB

The WriteDB function writes a database record containing PLC data structures.

Syntax

```
ret_code := WriteDB (VarName, RecordNum, DbHandle, DbType, Mode) ;
```

Input parameters

<i>VarName</i> (STRING)	PLC variable name to be written in the database record
<i>RecordNum</i> (INT)	data base record number to write
<i>DbHandle</i> (DINT)	database handle
<i>DbType</i> (INT)	retentive memory or memory mapped file
<i>Mode</i> (INT)	defines what to do if the PLC variable type does not really match with the data structures type that can be stored in the database.

Execution mode

Wait

Use

The function allows to write a PLC data structure (contained in the PLC variable *VarName*) in a database record. The database where to write in is identified by the handle, defined in the *DbHandle* parameter (OpenDB function output parameter) and by the fact that the Database is stored in a retentive memory or in a memory mapped file (*DbType* parameter).

The database record where to write in is identified by the record number (*RecordNum* parameter); the *RecordNum* parameter can have values from 1 (first database record)to the number of records in the database. The number of records in the database is defined when creating the database through the function CreateDB, and can be changed using ResizeDB function.

These conditions must exist for the correct record writing in the database:

- ▷ The database to write must exist, therefor it has to be previously created using the CreateDB function.
- ▷ The database to write must be opened, therefore it has to be previously opened using the OpenDB function that returns the handle identifying the database.
- ▷ The record where to write in must exist

The *VarName* variable must be structure type.

The *VarName* variable can be one of the following variables:

- ▷ A global variable: the variable can be identified indicating the variable name in the *VarName* parameter, i.e. 'GlobalVarName'
- ▷ A local variable defined within a program: the variable can be identified writing the program instance name, followed by the variable name, in the *VarName* parameter, i.e. 'ProgramInstanceName.VarName'
- ▷ A local variable defined within a function block: the variable can be identified can be identified writing in the *VarName* parameter the program instance name followed by the function block instance name, followed by the the variable name, i.e. 'ProgramInstanceName.FunctionBlockInstanceName.VarName'

The following table list all possible values *DbType* parameter can have:

DbType	Mnemonic	Description
0	dbRetainMemory	The database is stored in a retentive memory
1	dbMemoryMappedFile	The database is stored in a memory mapped file

The following table describes all possible values *Mode* parameter can have:

Mode	Description
0	If the VarName variable type is different from data structures that can be stored in the database, the writing doesn't occur; the database record does not change
1	If the VarName variable type is different from the data structures that can be stored in the database, only the fields having same name and type are written.

Return values

The following table lists all possible values *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x003C0001	Unavailable retentive memory
0x003C0002	Error in the database name; usually the name is too long
0x003C0003	The database to create already exists
0x003C0004	The retentive memory does not have space enough to store the database to create
0x003C0005	Internal error
0x003C0006	Error in the memory mapped file creation
0x003C0007	Error in the memory mapped file writing
0x003C0008	Error in the DbType parameter value
0x003C0014	VarName parameter value error: usually the parameter contains the empty string
0x003C0015	The VarName does not exists within the PLC program

Example

```

Ret      : dword;
DbHandle : dint;
...
(* writes the data structure contained in the PLC global variable named GlbVar
in record 2 of the database identified by the DbHandle handle in the retentive
memory *)

```

```

Ret := WriteDB('GlbVar', 2, DbHandle, dbRetainMemory, 1);

```

See also

CreateDB, OpenDB, ReadDB

11. Communication using TCP-IP and UDP Socket

SocketAccept	Accepts an input connection request
SocketClose	Closes the socket
SocketCreate	Creates a socket
SocketListen	Sets a socket in connection requests listening status
SocketSelect	Restores the socket status
SocketBind	Binds a local address to a socket
SocketConnect	Tries to set a connection towards a certain destination address
SocketEnableNonBlocking	Enables/disables a non-blocking mode on a socket
SocketReceive	Receives data from a socket
SocketSend	Sends data

11.1 SocketAccept

The `SocketAccept` function accepts an inbound connection request.

Syntax

```
ret_code := SocketAccept (ListeningSocket, AcceptSocket) ;
```

Input parameter

ListeningSocket (UDINT) Socket already listening for connection requests

Output parameter

AcceptSocket (UDINT) Socket created by the function that will be used to manage the connection

Use

The `SocketAccept` function allows to accept an inbound connection request.

The function releases the first connection request in the connection requests queue of the socket *ListeningSocket* socket, creating a new socket *AcceptSocket*; the *AcceptSocket* socket can be used to manage the connection, which is to send and receive data on the connection.

If the connections request queue of the *ListeningSocket* socket is empty, then:

- ▷ If *ListeningSocket* is in blocking mode, the function `SocketAccept` stops until the connection request is queued
- ▷ if *ListeningSocket* is in non-blocking mode, the `SocketAccept` function displays error

When `SocketAccept` function exits, the *ListeningSocket* socket waits for connection requests.

In case of TCP protocol, the `SocketAccept` function is usually used by a server that can have one or more connection requests at the same time.

In order to accept the connection, a server must:

- ▷ create a socket using the `SocketCreate` function
- ▷ bind it to a local address through the `SocketBind` function
- ▷ set it in connection requests listening mode using the `SocketListen` function
- ▷ accept the connection through the function `SocketAccept`

Only now the connection is enabled to send and receive data

In case of UDP protocol the `SocketAccept` function must not be called.

Return values

Value (HEX)	Description
0	Function executed without errors
0x0039001E	Network out of order
0x00390008	No buffer space available or too many connection requests
0x0039000D	If socket <i>ListeningSocket</i> socket has not be set in connection requests listening mode using the <i>SocketListen</i> function
0x00390007	The <i>ListeningSocket</i> parameter is an invalid socket. <i>SocketCreate</i> could have not be called or couldn't have been successful
0x0039000C	The type of <i>ListeningSocket</i> socket does not allow to accept connection requests
0x0039000F	If socket <i>ListeningSocket</i> socket is in non-blocking mode and there are no connection requests queued
0x00390020	General error; this error should never occur

Example

```

VAR
create_res, bind_res, listen_res:  DWORD;
Socket : UDINT;
AcceptSocket : UDINT;
SocketAddress : SocketAddressType;

END_VAR

create_res := SocketCreate(SCK_AF_IPv4, SCK_PROTO_TCP, Socket);

if (create_res = 0) then
    SocketAddress.PortNumber := 34999;
    SocketAddress.Address := '';
    bind_res := SocketBind(Socket, SocketAddress);
    if (bind_res = 0) then
        listen_res := SocketListen(Socket, 10);
        if (listen_res = 0) then
            accept_res := SocketAccept(Socket, AcceptSocket);
        end_if;
    end_if;
end_if;

```

11.2 SocketClose

The `SocketClose` closes the socket.

Syntax

```
ret_code := SocketClose (Socket) ;
```

Input parameter

<code>Socket</code> (UDINT)	Socket to close
-----------------------------	-----------------

Use

The `SocketClose` function closes a socket. Once socket is closed it cannot be used anymore.

Every socket created by the application through the `SocketCreate` function, must be closed by the `SocketClose` function when no more needed.

Return values

alue (HEX)	Description
0	Function executed without errors
0x0039001E	Network out of order
0x00390007	The <code>Socket</code> parameter is an invalid socket. <code>SocketCreate</code> could have not been called or couldn't have been successful
0x00390020	General error; this error should never occur

Example

```
VAR
close_res:  DWORD ;
Socket : UDINT ;

END_VAR

close_res := SocketClose(Socket);
```

11.3 SocketCreate

The **SocketCreate** function Creates a socket.

Syntax

```
ret_code := SocketCreate (AddressFamily, ProtocolType, Socket) ;
```

Input parameters

<i>AddressFamily</i> (DINT)	Addresses family
<i>ProtocolType</i> (DINT)	Protocol type

Output parameter

<i>Socket</i> (UDINT)	Socket created by the function
-----------------------	--------------------------------

Use

The **SocketCreate** function Creates a socket that can be used to send/receive information on a net using either the TCP or the UDP protocol.

The *AddressFamily* parameter can have following values:

AddressFamily	Value	Description
SCK_AF_IPv4	0	Internet Protocol version 4 (IPv4)

The *ProtocolType* parameter can have following values:

ProtocolType	Value	Description
SCK_PROTO_TCP	1	Transmission Control Protocol (TCP)
SCK_PROTO_UDP	2	User Datagram Protocol (UDP)

A socket is created in blocking mode by default.

If you want to use the socket also in non-blocking mode, after creating the socket you have to call the **SocketEnableNonblocking** primitive.

Return values

Value (HEX)	Description
0	Function executed without errors
0x00390001	Invalid <i>AddressFamily</i> parameter
0x00390002	Invalid <i>ProtocolType</i> parameter
0x0039001E	Network out of order
0x00390003	Too many sockets used at the same time
0x00390008	No buffer space available
0x00390009	Type of socket not supported
0x00390020	General error; this error should never occur

Example

```
VAR  
  Ret_code:  DWORD ;  
  Socket : UDINT ;  
  
END_VAR  
  
Ret_code := SocketCreate(SCK_AF_IPv4, SCK_PROTO_TCP, Socket);
```

11.4 SocketListen

The **SocketListen** function places a socket in listening status for connection requests.

Syntax

```
ret_code := SocketListen (Socket, SocketAddress) ;
```

Input parameter

<i>Socket</i> (UDINT)	Socket to place in listening status for connection requests
<i>Backlog</i> (DINT)	Maximum number of connections that can be queued

Use

The **SocketListen** function allows to place a socket in listening status for connection requests; after calling this function, connection requests will be placed in queue waiting to be accepted. The maximum number of connections to place in queue is defined by the *Backlog* parameter.

In case of TCP protocol, the **SocketListen** function is usually used by a server that can have one or more connection requests at the same time.

In order to accept the connection, a server must:

- ▷ create a socket using the **SocketCreate** function
- ▷ bind it to a local address through the **SocketBind** function
- ▷ set it in connection requests listening mode using the **SocketListen** function
- ▷ accept the connection through the function **SocketAccept**

Only now the connection is enabled to send and receive data

In case of UDP protocol the **SocketListen** function must not be called.

Return values

Value(HEX)	Description
0	Function executed without errors
0x0039001E	Network out of order
0x00390008	Too many connections
0x00390004	A process in the computer is already using the local address <i>SocketAddress</i>
0x0039000A	The <i>Socket</i> socket is not associated to a local address through SocketBind function.
0x0039000B	The <i>Socket</i> socket is already connected
0x00390007	The <i>Socket parameter</i> is an invalid socket. SocketCreate could have not been called or couldn't be successful.
0x0039000C	The socket type does not allow to be placed in listening mode for connection requests.
0x00390020	General error; this error should never occur

Example

```
VAR
create_res, bind_res, listen_res:  DWORD ;
Socket : UDINT ;
SocketAddress : SocketAddressType;

END_VAR

create_res := SocketCreate(SCK_AF_IPv4, SCK_PROTO_TCP, Socket);

if (create_res = 0) then
    SocketAddress.PortNumber := 34999;
    SocketAddress.Address := '';
bind_res := SocketBind(Socket, SocketAddress);
if (bind_res = 0) then
    listen_res := SocketListen(Socket, 10);
end_if;
end_if;
```

11.5 SocketSelect

The `SocketSelect` function returns a socket status.

Syntax

```
ret_code := SocketSelect (Socket, Timeout, Status) ;
```

Input parameters

<code>Socket</code> (UDINT)	Socket
<code>Timeout</code> (INT)	Timeout, in ms

Output parameter

<code>Status</code> (UINT)	Socket status: read, write or in error
----------------------------	--

Use

The `SocketSelect` function returns the `Socket` socket status.

The `Timeout` parameter, in ms, indicates the maximum execution time of the `SocketSelect` function. If the value of `Timeout` parameter is lower than zero, the function stops until the socket status becomes available. If the value of `Timeout` parameter is equal to zero, the function returns immediately.

The `Status` output parameter is made by different bits; each bit has the following meaning:

Bit	Description
0x0001	Socket can be read
0x0002	Socket can be written
0x0004	Socket in error

The `SocketSelect` function must be called after a call to the `SocketConnect` function on a `Socket` in non-blocking mode to check if the connection attempt was successful or not.

Following a call to the `SocketConnect` function in non-blocking mode, the connection attempt usually does not stop immediately. The `SocketConnect` function returns the error `0x0039000F`, but the connection attempt does not stop. To check whether the connection attempt was successful or not, the function `SocketSelect` should be called more than once checking if the socket is writable: if the socket can be written, the connection attempt was successful and the socket can be used to send/receive data on the connection.

Return values

Value(HEX)	Description
0	Function executed without errors
0x0039001E	Network is not working
0x00390007	The <code>Socket</code> parameter is an invalid socket. The <code>SocketCreate</code> could not have been called or could not have been successful
0x00390020	General error; this error should never occur

Example

```
VAR
    Socket: UDINT;
    create_res: DWORD := 100;
    connect_res: DWORD := 100;
    nonblocking_res: DWORD := 100;
    select_res: DWORD := 100;
    NonblockingMode: BOOL := true;
    bConnectionUnderWay: BOOL := false;
    DestinationAddress : SocketAddressType;
    DestinationIpAddress : string := 'pagace';
    bConnectionOk : bool := false;
    Timeout : INT := 100;
    Status : uint := 0;

END_VAR
    create_res := SocketCreate(SCK_AF_IPv4, SCK_PROTO_TCP, Socket);
    nonblocking_res := SocketEnableNonblocking(Socket, NonblockingMode);

    DestinationAddress.PortNumber := 34999;
    DestinationAddress.Address := DestinationIpAddress;
    connect_res := SocketConnect(Socket, DestinationAddress);
    if connect_res = 0 then
        bConnectionOk := true;
    else
        if (NonblockingMode) then
            bConnectionUnderWay := true;
        end_if;
    end_if;

    if (bConnectionUnderWay) then
        select_res := SocketSelect(Socket, Timeout, Status);
        if Status > 0 then
            bConnectionUnderWay := false;
        end_if;
        if Status = 2 then
            bConnectionOk := true;
        end_if;
    end_if;
```

11.6 SocketBind

The **SocketBind** function binds a local address to a socket.

Syntax

```
ret_code := SocketBind (Socket, SocketAddress) ;
```

Input parameters

<i>Socket</i> (UDINT)	Socket
<i>SocketAddress</i> (SocketAddressType)	Local address

Use

The **SocketBind** function allows to bind a local address to a socket.

The socket passed in the *Socket* parameter has to be already created using the **SocketCreate** function.

The *SocketAddress* parameter is a structure allowing to specify the local address to bind to the socket. The *SocketAddressType* structure is made by following fields:

Field	Type	Description
PortNumber	UDINT	Port number
Address	STRING	IP address

The *SocketAddress* parameter is useful if the machine has more than one network board. If for the application is not important the local address to use, or if the machine has only one network board, the field *Address* of the structure can be set on the empty string.

The **SocketBind** function is typically used as follow:

- ▷ In case of TCP protocol, from server side, after creating the socket and before setting it in connection attempts listening using the primitive **SocketListen**

- ▷ In case of UDP protocol, in receipt phase, after creating the socket and before trying to get a message using the **SocketReceive** primitive

Return values

Value (HEX)	Description
0	Function executed without errors
0x0039001E	Network out of order
0x00390008	No buffer space available
0x00390004	A process in the computer is already using the local address <i>SocketAddress</i>
0x00390005	The <i>SocketAddress</i> is an invalid address in this computer
0x00390006	The <i>Socket</i> socket is already bound to an address
0x00390007	The <i>Socket</i> parameter is an invalid socket. The SocketCreate could not have been called or could not have been successful
0x00390020	General error; this error should never occur

Example

```
VAR
create_res, bind_res:  DWORD ;
Socket : UDINT ;
SocketAddress : SocketAddressType;

END_VAR

create_res := SocketCreate(SCK_AF_IPv4, SCK_PROTO_TCP, Socket);

if (create_res = 0) then
    SocketAddress.PortNumber := 34999;
    SocketAddress.Address := '';
    bind_res := SocketBind(Socket, SocketAddress);
end_if;
```

11.7 SocketConnect

The `SocketConnect` function tries to set a connection towards a certain destination address.

Syntax

```
ret_code := SocketConnect (Socket, SocketAddress) ;
```

Input parameters

<code>Socket</code> (UDINT)	Socket for the connection
<code>SocketAddress</code> (<code>SocketAddressType</code>)	Address towards which set the connection

Use

The `SocketConnect` tries to set a connection towards a certain destination address. If the function is successful, the socket `Socket` can be used to send/receive data on the connection. The socket passed in the `Socket` parameter has to be created using the function `SocketCreate`.

The `SocketAddress` parameter is a structure allowing to specify the destination address towards which the connection has to be set. The `SocketAddressType` structure is made by following fields:

Field	Type	Description
<code>PortNumber</code>	UDINT	Port number
<code>Address</code>	STRING	IP address

If the `Socket` socket is in blocking mode, the return value of the function shows if the connection attempt was successful or not.

If `SocketConnect` socket is in non-blocking mode, the connection attempt usually does not stop immediately: in this case the function returns the error `0x0039000F`, but the connection attempt does not stop. To check whether the connection attempt was successful or not, the function `SocketSelect` should be called more than once checking if the socket is writable: if the socket can be written, the connection attempt was successful and the socket can be used to send/receive data on the connection.

The `SocketConnect` function is usually used in case of TCP protocol from a client.

To allow a client in setting a connection, it must:

- ▷ Create a socket using the `SocketCreate` function
- ▷ Try to set a connection using the `SocketConnect` function

When server accepts the connection request, data on the connection can be sent/received.

In case of UDP protocol the function `SocketConnect` does not have to be called.

Return values

Value (HEX)	Description
0	Function executed without errors
0x0039001E	Network out of order
0x00390010	Socket is in listening mode for connection attempts
0x00390008	Socket cannot be connected
0x00390007	The <i>Socket</i> parameter is an invalid socket. The <i>SocketCreate</i> could not have been called or could not have been successful
0x0039000F	The <i>Socket</i> socket is in non-blocking mode and the connection attempt is not stopped yet.
0x00390011	A previous non-blocking call to <i>SocketConnect</i> is already running on the socket <i>Socket</i>
0x00390005	Invalid remote address
0x00390012	Connection attempt rejected
0x00390013	Wrong remote address
0x0039001B	Socket already connected
0x00390014	Timeout expired on the connection attempt
0x00390020	General error; this error should never occur

Example

```

VAR
create_res, connect_res: DWORD ;
Socket : UDINT ;
DestinationAddress: SocketAddressType;

END_VAR

create_res := SocketCreate(SCK_AF_IPv4, SCK_PROTO_TCP, Socket);

if (create_res = 0) then
  DestinationAddress.PortNumber := 34999;
  DestinationAddress.Address := '172.30.2.31';
  connect_res := SocketConnect(Socket, DestinationAddress);
end_if;

```

11.8 SocketEnableNonblocking

The `SocketEnableNonblocking` function enables/disables the non-blocking mode on a socket.

Syntax

```
ret_code := SocketEnableNonblocking (Socket, NonblockingMode) ;
```

Input parameters

<i>Socket</i> (UDINT)	Socket
<i>NonblockingMode</i> (BOOL)	true to enable non-blocking mode false to disable non-blocking mode

Use

The `SocketEnableNonblocking` function enables/disables the non-blocking mode on the socket *Socket*.

If the *NonblockingMode* parameter is true, the function `SocketEnableNonblocking` enables the non-blocking mode. If the *NonblockingMode* parameter is false, the `SocketEnableNonblocking` function disables the non-blocking mode.

Note that a socket is created in blocking mode by default.

If you want to use the socket in non-blocking mode, following the socket creation, you have to call the `SocketEnableNonblocking` function setting true as value of the *NonblockingMode* parameter.

Return values

Value (HEX)	Description
0	Function executed without errors
0x0039001E	Network out of order
0x00390007	The <i>Socket</i> parameter is an invalid socket. The <code>SocketCreate</code> could not have been called or could not have been successful
0x00390020	General error; this error should never occur

Example

VAR

```
create_res: DWORD;
nonblocking_res: DWORD;
Socket : UDINT ;
```

END_VAR

```
create_res := SocketCreate(SCK_AF_IPv4, SCK_PROTO_TCP, Socket);
nonblocking_res := SocketEnableNonblocking(Socket, true);
```

11.9 SocketReceive

The **SocketReceive** function receives data from a socket.

Syntax

```
ret_code := SocketReceive (Socket, Buffer, ReceivedDataLen) ;
```

Input parameter

<i>Socket</i> (UDINT)	Socket to use for receiving data
-----------------------	----------------------------------

Output parameters

<i>Buffer</i> (ARRAY OF BYTE)	Buffer used by the function to store data received
<i>ReceivedDataLen</i> (DINT)	Length (in byte) of data received by the function

Use

The **SocketReceive** function receives data from a socket and can be used for both the TCP and the UDP protocol. In case of TCP protocol, before receiving the data, a connection must be set. Moreover, in case of TCP protocol, the **SocketReceive** function will return all data received so far, up to the *Buffer* parameter dimension.

In case of UDP protocol, before receiving data, a socket has to be bound to a local address calling the **SocketBind** function. If the dimension of data received exceeds the *Buffer* parameter dimension, the *Buffer* will be filled with the starting part of the data and the function **SocketReceive** returns the error 0x00390018.

In case of UDP, data not in the buffer will be lost; instead, in TCP case, these data are kept and will be returned to the next **SocketReceive** function call.

If data arrived, then:

- If the *Socket* socket is in blocking mode, the function **SocketReceive** stops until data arrive
- If the *Socket* socket is in non-blocking mode, the function **SocketReceive** returns error having 0x0039000F return value

If no data arrived, the function writes value 0 in the first byte of the receipt buffer (*Buffer* parameter).

Return values

Value (HEX)	Description
0	Function executed without errors
0x0039001E	Network out of order
0x00390007	The <i>Socket</i> parameter is an invalid socket. The SocketCreate could not have been called or could not have been successful
0x0039000F	The <i>Socket</i> socket is in non-blocking mode and in this moment no data have been received
0x00390014	Connection interrupted as the network does not work or because the remote system crashed
0x00390015	Connection interrupted because of an error when executing the operation.
0x00390016	Socket is not connected
0x00390017	Socket is closed

Value (HEX)	Description
0x00390018	The message exceeds the receiving buffer dimension
0x0039001A	Connection aborted. Socket cannot be used anymore
0x0039001B	Connection closed by remote system. Socket cannot be used anymore
0x0039000A	The <i>Socket</i> socket was not bound to a local address using the <i>SocketBind</i> function
0x00390020	General error; this error should never occur

Example

```

VAR
  UdpSocket: UDINT;
  create_res: DWORD;
  bind_res: DWORD;
  receive_res: DWORD;
  nonblocking_res: DWORD;
  RxBuffer : array[0..99] of byte;
  ReceivedDataLen : DINT;
  SocketAddress : SocketAddressType;
  SuccessfulRxNum : DINT;
END_VAR

create_res := SocketCreate(SCK_AF_IPv4, SCK_PROTO_UDP, UdpSocket);
if (create_res = 0) then
  nonblocking_res := SocketEnableNonblocking(UdpSocket, true);
  SocketAddress.PortNumber := 34900;
  SocketAddress.Address := '';
  bind_res := SocketBind(UdpSocket, SocketAddress);
end_if;

if (bind_res = 0) then
  receive_res := SocketReceive(UdpSocket, RxBuffer, ReceivedDataLen);
  if receive_res = 0 then
    SuccessfulRxNum := SuccessfulRxNum + 1;
  end_if;
end_if;

```

11.10 SocketSend

The SocketSend function sends data.

Syntax

```
ret_code := SocketSend (Socket, Data, DataLen, DestinationAddress, SentDataLen) ;
```

Input parameters

<i>Socket</i> (UDINT)	Socket to use for data sending
<i>Data</i> (ARRAY OF BYTE)	Data to send
<i>DataLen</i> (DINT)	Length (in byte) of data to send
<i>DestinationAddress</i> (SocketAddressType)	Address towards which the data has to be send, important only for the UDP protocol

Output parameter

<i>SentDataLen</i> (DINT)	Length (in byte) of the data sent by the function
---------------------------	---

Use

The SocketSend function sends data and can be used for both the TCP and the UDP protocol. In case of TCP protocol, before sending data, a connection must be set. Moreover, in case of TCP protocol, the *DestinationAddress* parameter is not important. In case of UDP protocol, the *DestinationAddress* parameter, SocketAddressType structure type, is important: it is the address towards which sending data.

The SocketAddressType structure is made by following fields:

Field	Type	Description
PortNumber	UDINT	Port number
Address	STRING	IP address

Even if the SocketSend function has been successfully executed, the data could not have been received by the recipient, but sent with success.

The transport system does not have enough space to store all data to be transmitted:

- ▷ If *Socket* socket is in blocking mode, the function SocketSend stops until the space is free.
- ▷ If *Socket* socket is in non-blocking mode, the function SocketSend could return an error having return value = 0x0039000F or could be successful, but the length of data sent (*SentDataLen* parameter) can be higher than zero, but lower than the data length to send (*DataLen* parameter).

Return values

Value (HEX)	Description
0	Function executed without errors
0x0039001E	Network out of order
0x00390008	Unavailable space
0x00390007	The <i>Socket</i> parameter is an invalid socket. The <i>SocketCreate</i> could not have been called or could not have been successful
0x0039000F	The <i>Socket</i> socket is in non-blocking mode and in this moment no data can be transmitted
0x00390005	Invalid remote address
0x00390013	Wrong remote address
0x00390014	Connection interrupted because the network does not work or because remote system crashed
0x00390015	Connection interrupted because of an error when executing the operation
0x00390016	Socket is not connected
0x00390017	Socket is closed
0x00390018	Message exceeds the maximum allowed
0x00390019	In this moment remote system cannot be reached
0x0039001A	Connection aborted. Socket cannot be used anymore
0x0039001B	Connection closed by remote system. Socket cannot be used anymore
0x0039001C	Destination address is necessary (<i>DestinationAddress</i> parameter)
0x0039001D	Network cannot be reached at the moment
0x00390020	General error; this error should never occur

Example

```

VAR
  UdpSocket: UDINT;
  create_res: DWORD;
  send_res: DWORD;
  nonblocking_res: DWORD;
  NonblockingMode: BOOL := true;
  Data : array[0..4] of byte;
  SentDataLen : DINT;
  DestinationAddress : SocketAddressType;
END_VAR

create_res := SocketCreate(SCK_AF_IPv4, SCK_PROTO_UDP, UdpSocket);
nonblocking_res := SocketEnableNonblocking(UdpSocket, NonblockingMode);

Data[0] := 10;
Data[1] := 11;
Data[2] := 12;
Data[3] := 13;
Data[4] := 14;
DestinationAddress.PortNumber := 34900;
DestinationAddress.Address := '172.30.2.31';

send_res := SocketSend(UdpSocket, Data, 5, DestinationAddress, SentDataLen);

```

12. Serial line related functions - 4C_SERIALCOMM

SerialLineClose	Closes a work session for the serial line indicated in input
SerialLineOpen	Opens a work session for the serial line indicated in input
SerialLineReadSetup	Reads the configuration of a serial line
SerialLineWriteSetup	Is used to configure a serial line
SerialLineReceive	Executes the reception of characters via the serial line specified
SerialLineTransmit	Transmits characters via the serial line specified
SerialLineReset	Function stops the activity of a serial line and resets the error condition
SerialLineStatus	Reads the status of a serial line

12.1 SerialLineClose

The SerialLineClose function closes a work session for the serial line indicated as input.

Syntax:

```
ret_code := SerialLineClose (ExecMode, ExecStatus, Line) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>Line</i> (INT)	Serial line number [1..4]

Output parameters:

<i>ExecStatus</i> (DWORD)	Execution status of command in MODE_NOWAIT_ACK mode
---------------------------	---

Execution mode:

Defined by *ExecMode*

Use:

This function is used to close a work session with a serial line when you do want to use it any longer. The line must be in s_CONNECTED status, and when the session has been closed, its status changes to s_DISCONNECTED. To reopen a work session, use the SerialLineOpen function.

Return values:

The following table lists all possible values *ret_code* variable can have:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00310002	Line value out of range
0x00310003	Serial line not configured or not found in system
0x00310004	Serial line in status not compatible with request
0x00310005	ExecMode value out of range
0x00310021	Error in closing the physical device

Example:

```
EXEC_STATUS at GL40 : dword;
Ret : dword;
(*COM1 serial line closing in no wait mode.*)
Ret := SerialLineClose (MODE_NOWAIT, EXEC_STATUS, 1);
```

See also:

[SerialLineOpen](#)

12.2 SerialLineOpen

The SerialLineOpen function opens a work session for the serial line indicated in input.

Syntax:

```
ret_code := SerialLineOpen (ExecMode, ExecStatus, Line) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>Line</i> (INT)	Serial line number [1..4]

Output parameters:

<i>ExecStatus</i> (DWORD)	Execution status of command in MODE_NOWAIT_ACK mode
---------------------------	---

Execution mode:

Defined by *ExecMode*

Use:

This function is used to open a work session with a serial line, i.e., to be able to send and receive characters. The line must be in s_DISCONNECTED status, and once the session has been opened, line status changes to s_CONNECTED. To close a work session, use the SerialLineClose function.

Return values:

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00310002	Line value out of range
0x00310003	Serial line not configured or not found in system
0x00310004	Serial line in status not compatible with request
0x00310005	Value of <i>ExecMode</i> out of range
0x00310020	Error in opening the physical device
0x00310023	Error in parameterisation of the physical device
0x00310025	Error in setting timeouts on physical device
0x00310027	Error in cleaning the physical device
0x00310028	Error in setting events of physical device

Example:

```
EXEC_STATUS at GL40 : dword;
Ret : dword;
(*COM1 serial line opening in wait mode.*)
Ret := SerialLineOpen (MODE_WAIT, EXEC_STATUS, 1);
```

See also:

SerialLineClose

12.3 SerialLineReadSetup

The `SerialLineReadSetup` function reads the configuration of a serial line.

Syntax:

ret_code := SerialLineReadSetup (*Line* , *Setup*);

Input parameters:

Line (INT) Serial line number [1..4]

Output parameters:

Setup (SerialLineCFG_struct) Serial line parameters

Execution mode:

Immediate

Use:

This function lets you read the parameterisation of a serial line.

Following lines describe the meaning of the *Setup* parameter structure fields:

Field	Type	Description
BaudRate	DWORD	Transmission rate
Parity	WORD	Parity control type
ByteSize	WORD	Received/sent byte size
StopBits	WORD	Stop bits number
Protocol	WORD	Protocol type
FlowCtrl	DWORD	Flow control
TXTimeout	WORD	Transmission timeout in milliseconds, if 0 disabled
TXRetry	WORD	Number of transmission repetition attempts in case of error
RXTimeout	WORD	Reception timeout in milliseconds, if 0 disabled
RXRetry	WORD	Number of reception repetition attempts in case of error
Terminator	WORD	Receipt character finished (see also FlowCtrl)
Reserved1	DWORD	Filed reserved, set to 0
Reserved2	DWORD	Filed reserved, set to 0

For each file the following definition is ok:

BaudRate	Mnemonic	Description
50	s_BAUD_50	Baudrate a 50 bps
75	s_BAUD_75	Baudrate a 75 bps
110	s_BAUD_110	Baudrate a 110 bps
134	s_BAUD_134	Baudrate a 1345 bps
150	s_BAUD_150	Baudrate a 150 bps
300	s_BAUD_300	Baudrate a 300 bps
600	s_BAUD_600	Baudrate a 600 bps
1200	s_BAUD_1200	Baudrate a 1200 bps
1800	s_BAUD_1800	Baudrate a 1800 bps

2000	s_BAUD_2000	Baudrate a 2000 bps
2400	s_BAUD_2400	Baudrate a 2400 bps
3600	s_BAUD_3600	Baudrate a 3600 bps
4800	s_BAUD_4800	Baudrate a 4800 bps
7200	s_BAUD_7200	Baudrate a 7200 bps
9600	s_BAUD_9600	Baudrate a 9600 bps
14400	s_BAUD_14400	Baudrate a 14400 bps
19200	s_BAUD_19200	Baudrate a 19200 bps
38400	s_BAUD_38400	Baudrate a 38400 bps
56000	s_BAUD_56000	Baudrate a 56000 bps
57600	s_BAUD_57600	Baudrate a 57600 bps
115200	s_BAUD_115200	Baudrate a 115200 bps
128000	s_BAUD_128000	Baudrate a 128000 bps
256000	s_BAUD_256000	Baudrate a 256000 bps

Parity	Mnemonic	Description
0	s_NOPARITY	No parity
1	s_ODDPARITY	Odd parity
2	s_EVENPARITY	Even parity
3	s_MARKPARITY	Mark parity
4	s_SPACEARITY	Space parity

ByteSize	Mnemonic	Description
5	s_BYTE5	5 bits transmission
6	s_BYTE6	6 bits transmission
7	s_BYTE7	7 bits transmission
8	s_BYTE8	8 bits transmission

StopBits	Mnemonic	Description
0	s_ONESTOPBIT	Un bit di stop
1	s_ONE5STOPBIT	Un bit e mezzo di stop
2	s_TWOSTOPBIT	Due bit di stop

Protocol	Mnemonic	Description
0	s_NORMAL	Normal
1	s_XONXOFF	Xon Xoff

FlowCtrl (bit)	Mnemonic	Description
0x0001	s_DTR	DTR enabled
0x0002	s_DTRHANDSHAKE	DTR HandShake
0x0004	s_RTS	RTS enabled
0x0008	s_RTSHANDSHAKE	RTS HandShake
0x0010	s_CTS	CTS management enabled
0x0020	s_DSR	DSR management enabled
0x0040	s_DSRSENSITY	Receipt ignored if DSR off
0x0080	s_CONTINUEXONXOFF	Xon Xoff continuous
0x0100	s_FOUTX	Xoff management
0x0200	s_FINX	Xon management

0x0400	s_TERMINATOR	Terminator management in reception
0x8000	s_FREEAutoRTSchar	Reception cleaning buffer after RTS off (valid only for CE5.0)

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00310002	Line value out of range
0x00310003	Serial line not configured or not found in system

Example:

```

Ret : dword;
SlcCfg : SerialLineCFG_struct;
Ret := SerialLineReadSetup (1, SlcCfg);

COM1 serial line parameter reading.

```

See also:

[SerialLineWriteSetup](#).

12.4 SerialLineWriteSetup

The SerialLineWriteSetup function is used to configure a serial line.

Syntax:

```
ret_code := SerialLineWriteSetup ( Line , Setup ) ;
```

Input parameters:

<i>Line</i> (INT)	Serial line number [1..4]
<i>Setup</i> (SerialLineCFG_struct)	Serial line parameters

Execution mode:

Immediate

Use:

This function lets you parameterise a serial line when the latter is in s_DISCONNECTED status (session not open). The parameters are updated when the next line opening command is given.

The meaning of the fields of the *Setup* parameter structure is listed below.

Field	Type	Description
BaudRate	DWORD	Transmission feed
Parity	WORD	Parity control type
ByteSize	WORD	Dimension of the character sent/received
StopBits	WORD	Stop bits number
Protocol	WORD	Protocol type
FlowCtrl	DWORD	Flow control
TXTimeout	WORD	Transmission timeout in milliseconds, if 0 disabled
TXRetry	WORD	Number of transmission repetition attempts in case of error
RXTimeout	WORD	Reception timeout in milliseconds, if 0 disabled
RXRetry	WORD	Number of reception repetition attempts in case of error
Terminator	WORD	Terminator receipt character (see also FlowCtrl)
Reserved1	DWORD	Filed reserved, set to 0
Reserved2	DWORD	Filed reserved, set to 0

For each fields is valid the following definition

BaudRate	Mnemonic	Description
50	s_BAUD_50	50 bps Baudrate
75	s_BAUD_75	75 bps Baudrate
110	s_BAUD_110	Baudrate at 110 bps Baudrate
134	s_BAUD_134	Baudrate at 1345 bps Baudrate
150	s_BAUD_150	Baudrate at 150 bps Baudrate
300	s_BAUD_300	Baudrate at 300 bps Baudrate
600	s_BAUD_600	Baudrate at 600 bps Baudrate
1200	s_BAUD_1200	Baudrate at 1200 bps Baudrate
1800	s_BAUD_1800	Baudrate at 1800 bps Baudrate

2000	s_BAUD_2000	Baudrate at 2000 bps Baudrate
2400	s_BAUD_2400	Baudrate at 2400 bps Baudrate
3600	s_BAUD_3600	Baudrate at 3600 bps Baudrate
4800	s_BAUD_4800	Baudrate at 4800 bps Baudrate
7200	s_BAUD_7200	Baudrate at 7200 bps Baudrate
9600	s_BAUD_9600	Baudrate at 9600 bps Baudrate
14400	s_BAUD_14400	Baudrate at 14400 bps Baudrate
19200	s_BAUD_19200	Baudrate at 19200 bps Baudrate
38400	s_BAUD_38400	Baudrate at 38400 bps Baudrate
56000	s_BAUD_56000	Baudrate at 56000 bps Baudrate
57600	s_BAUD_57600	Baudrate at 57600 bps Baudrate
115200	s_BAUD_115200	Baudrate at 115200 bps Baudrate
128000	s_BAUD_128000	Baudrate at 128000 bps Baudrate
256000	s_BAUD_256000	Baudrate at 256000 bps Baudrate

Parity	Mnemonic	Description
0	s_NOPARITY	No parity
1	s_ODDPARITY	Odd parity
2	s_EVENPARITY	Quits parity
3	s_MARKPARITY	Mark parity
4	s_SPACEARITY	Space parity

ByteSize	Mnemonic	Description
5	s_BYTE5	Transmission at 5 bit
6	s_BYTE6	Transmission at 6 bit
7	s_BYTE7	Transmission at 7 bit
8	s_BYTE8	Transmission at 8 bit

StopBits	Mnemonic	Description
0	s_ONESTOPBIT	One stop bit
1	s_ONE5STOPBIT	One and a half stop bit
2	s_TWOSTOPBIT	Two stop bits

Protocol	Mnemonic	Description
0	s_NORMAL	Normal
1	s_XONXOFF	Xon Xoff

FlowCtrl (bit)	Mnemonic	Description
0x0001	s_DTR	DTR enabled
0x0002	s_DTRHANDSHAKE	DTR HandShake
0x0004	s_RTS	RTS enabled
0x0008	s_RTSHANDSHAKE	RTS HandShake
0x0010	s_CTS	CTS management enabled
0x0020	s_DSR	DSR management enabled
0x0040	s_DSRSENSITY	Reception ignored if DSR off
0x0080	s_CONTINUEXONXOFF	Xon Xoff continuous
0x0100	s_FOUTX	Xoff management
0x0200	s_FINX	Xon management

0x0400	s_TERMINATOR	Reception terminator management
0x8000	s_FREEAutoRTSchar	Reception buffer cleaning after RTS off (valid only for CE5.0)

In particular :

bit	Description and possible values
Bit 0 .. 1	<i>DTR Control</i> , in combinations below (bit 0,1): 0,0 : DTR disabled 0,1 : DTR enabled in line opening 1,0 : reserved 1,1 : reserved
bit 2 .. 3	<i>RTS Control</i> , in combinations below (bit 3,2): 0,0 : RTS disabled 0,1 : RTS enabled in line opening 1,0 : reserved 1,1 : RTS at 1 in transmission start, at 0 in transmission end



For proper serial line operation, always set all the configuration fields before opening the line.

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00310002	Line value out of range
0x00310003	Serial line not configured or not found in system
0x00310004	Serial line in status not compatible with request
0x00310010	Invalid BaudRate value
0x00310011	Invalid ByteSize value
0x00310012	Invalid Parity value
0x00310013	Invalid FlowCtrl value
0x00310014	Invalid Protocol value
0x00310015	Invalid StopBits value

Example:

```
Ret : dword;
SlcCfg : SerialLineCFG_struct;
SlcCfg.BaudRate := 9600;
SlcCfg.StopBits := 1;
SlcCfg. ByteSize := 8;
SlcCfg. RXTimeout := 3000;
SlcCfg. FlowCtrl := 16#400;
SlcCfg. Terminator := 13;
```

COM1 serial line parameterisation. The example above shows the setup of several fields; in actual practice all the fields of the input structure have to be defined.

See also:

SerialLineReadSetup.

12.5 SerialLineReceive

The SerialLineReceive function executes the reception of characters via the serial line specified.

Syntax:

```
ret_code := SerialLineReceive (ExecMode, ExecStatus, Line, Buffer) ;
```

Input parameters:

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT]
<i>Line</i> (INT)	Serial line number [1..4]
<i>Length</i> (INT)	Number of characters required; it is important only if higher than 0.

Output parameters:

<i>ExecStatus</i> (DWORD)	Not used
<i>Length</i> (INT)	Number of characters received
<i>Buffer</i> (STRING)	Characters received [1..128]

Execution mode:

Defined through *ExecMode*

Use:

In MODE_NOWAIT mode, this function lets you read the characters received so far, including none, without any wait time (the number of characters received can be determined through the SerialLineStatus function). If terminator acknowledgement has been enabled, the function returns in *Buffer* the characters received up to and including the termination character.

If *Length* parameter has a value >0 when entering the function, the function returns a maximum number of characters equivalent to *Length*. The maximum number of characters to be returned is 128. It is not possible to execute a no wait reception if a wait is already in progress.

In MODE_WAIT mode the system checks if they are the proper conditions to receive (specified by the terminator or by the *Length* parameter) number of characters required, (number of character required, specified in the *Length* parameter, or terminator) and waits for their happening. Using the SerialLineWriteSetup function the receipt timeout can be set to stop the waiting in case the conditions do not happen at the time required; if timeout is set =0, the wait is infinite and can be interrupted only using the SerialLineReset function. In case of timeout, the system retries the receipt for the retry number set, in case of reception errors the wait will finish immediately.

During a reception, the main status of the line changes to s_ACTIVE and reception status changes to s_RX_RUN; at the end of the reception, if there are no other activities pending, line status goes back to s_CONNECTED and reception status to s_RX_IDLE.

The *Length* parameter contains the number of character received when exits the function.

Return values:

The following table lists all possible values *ret_code* variable can have

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00310002	Line value out of range
0x00310003	Serial line not configured or not found in system
0x00310004	Serial line in status not compatible with request
0x00310005	Value of ExecMode out of range
0x00310006	Buffer length exceeds 128 characters
0x0031002A	Error in cleaning physical device events
0x0031002B	Error in reading from physical device
0x00310030	Reception frames exhausted
0x00310032	Reception timeout
0x00310033	Reception overrun
0x00310034	Error in receiving physical device frame
0x00310035	Physical device parity error
0x00310036	Physical device reception overrun
0x00310037	Reading from physical device inconsistent

Example:

```
EXEC_STATUS at GL40 : dword;
Ret : dword;
Len : int
Buffer : string;

Len :=2
(Waits for reception of 2 characters from COM1 serial line in wait mode*).
Ret := SerialLineReceive (MODE_WAIT, EXEC_STATUS, 1, Buffer);
```

See also:

[SerialLineTransmit](#), [SerialLineWriteSetup](#)

12.6 SerialLineTransmit

The SerialLineTransmit function transmits characters via the serial line specified.

Syntax:

```
ret_code := SerialLineTransmit (ExecMode, ExecStatus, Line, Buffer) ;
```

Input parameters

<i>ExecMode</i> (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
<i>Line</i> (INT)	Serial line number [1..4]
<i>Buffer</i> (STRING)	Characters to send[1..128]

Output parameters:

<i>ExecStatus</i> (DWORD)	Execution status of command in MODE_NOWAIT_ACK mode
---------------------------	---

Execution mode:

Defined with *ExecMode*

Use:

This function is used to transmit characters via the serial line specified. During a transmission, the status of the line changes to s_ACTIVE and transmission status changes to s_TX_RUN; at the end of the transmission, if there are no other activities pending, line status goes back to s_CONNECTED and the sub-status changes back to s_TX_IDLE. You can queue several transmission frames without having to wait for them to end by calling the function in one of the no wait modes.

By setting bits 2 and 3 of the *FlowCtrl* field of the configuration structure to 1 you can enable the toggle management of the RTS signal, i.e., the signal logic signal is set to 1 at the start and 0 at the end of the transmission: If there are several transmission frames queued up, the signal level remains 2 until all frames have been transmitted, and during any retries as well.

Return values:

The following table lists all possible values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00310002	Line value out of range
0x00310003	Serial line not configured or not found in system
0x00310004	Serial line in status not compatible with request
0x00310005	ExecMode value out of range
0x00310006	Buffer length exceeds 128 characters
0x00310026	Error in writing to physical device
0x00310029	Error in setting RTS of physical device
0x00310030	Transmission frames exhausted

Example:

```
EXEC_STATUS at GL40 : dword;
Ret : dword;
Buffer : string;
Buffer := 'This is a string to transmit';

(*Transmission of a string via COM1 serial line in no wait mode.*)
Ret := SerialLineTransmit (MODE_NOWAIT, EXEC_STATUS, 1, Buffer);
```

See also:

[SerialLineReceive](#)

12.7 SerialLineReset

The SerialLineReset function stops the activity of a serial line and resets the error condition.

Syntax:

```
ret_code := SerialLineReset (ExecMode, ExecStatus, Line) ;
```

Input parameters:

ExecMode (INT)	Command execution mode [MODE_NOWAIT, MODE_WAIT, MODE_WAIT_ACCEPT, MODE_NOWAIT_ACK]
Line (INT)	Serial line number [1..4]

Output parameters:

ExecStatus (DWORD)	Execution status of command in MODE_NOWAIT_ACK mode
--------------------	---

Execution mode:

Defined by ExecMode

Use:

When this function is called, transmission and/or reception activities are stopped and a (recoverable) error condition the line may be in is cleaned. When reset operations are over, line status changes to s_CONNECTED and transmission and reception statuses change to idle.

Return values:

The following table lists all possible values that *ret_code* variable can have

<i>ret_code</i> (HEX)	Description
0x00000000	Function executed without errors
0x00310002	Line value out of range
0x00310003	Serial line not configured or not found in system
0x00310004	Serial line in status not compatible with request
0x00310005	Value of ExecMode out of range

Example:

```
Ret : dword;
EXEC_STATUS at GL40 : dword;

(Resets COM1 serial line in no wait mode*).
Ret := SerialLineReset (MODE_NOWAIT, EXEC_STATUS, 1);
```

12.8 SerialLineStatus

The SerialLineStatus function reads the status of a serial line.

Syntax:

```
ret_code := SerialLineStatus ( Line , Status ) ;
```

Input parameters:

<i>Line</i> (INT)	Serial line number [1..4]
-------------------	---------------------------

Output parameters:

<i>Status</i> (SerialLineSTS_struct)	Line status
--------------------------------------	-------------

Execution mode:

Wait

Use:

In the *Status* variable, this function returns the status of the serial line; the values of the individual fields of the structure are listed below.

Field	Type	Description
LineStatus	WORD	Serial line status
Error	DWORD	Main error code of the line
FreeTxFrames	WORD	Number of blocks free for transmission
TXStatus	WORD	Transmission status
TXError	DWORD	Transmission error code
FreeRxFrames	WORD	Number of blocks free for reception
RXStatus	WORD	Reception status
RXError	DWORD	Reception error code
RXCharRead	WORD	Number of characters received but not read yet
InternalSts	DWORD	Internal error status
Reserved1	DWORD	Reserved field
Reserved2	DWORD	Reserved field

LineStatus	Mnemonic	Description
0	s_NOTAVAIL	Line not available
1	s_DISCONNECTED	Line in closed session
2	s_CONNECTED	Line in opened session
3	s_ACTIVE	Line active in transmission and/or reception
4	s_INERROR	Unrecoverable error line

TxStatus	Mnemonic	Description
0	s_TX_IDLE	Disabled transmission
1	s_TX_RUN	Enabled transmission
2	s_TX_ERROR	Transmission error

RxStatus	Mnemonic	Description
0	s_RX_IDLE	Reception disabled
1	s_RX_RUN	Reception enabled
2	s_rX_ERROR	Reception error

Return values:

ret_code (HEX)	Description
0x00000000	Function executed without errors
0x00310002	Line value out of range
0x00310003	Serial line not configured or not found in system

Example:

```
Ret : dword;
SlcSts : SerialLineSTS_struct;

(*Reads status of COM1 serial line.*)
Ret := SerialLineStatus (1, SlcSts);
```

12.9 List of errors returned by Serial Line

Error code (HEX)	Description
0x00310001	Memory low
0x00310002	Invalid serial line number
0x00310003	Serial line not configured or not found in system
0x00310004	Line status not compatible with request
0x00310005	Invalid execution mode
0x00310006	Number characters exceeds admissible limit
0x00310010	Invalid configuration of BaudRate field
0x00310011	Invalid configuration of ByteSize field
0x00310012	Invalid configuration of Parity field
0x00310013	Invalid configuration of FlowCtrl field
0x00310014	Invalid configuration of Protocol field
0x00310015	Invalid configuration of StopBits field
0x00310020	Error in opening the physical device
0x00310021	Error in closing the physical device
0x00310022	Error in reading configuration of physical device
0x00310023	Error in writing configuration of physical device
0x00310024	Error in reading timeouts of physical device
0x00310025	Error in writing timeouts of physical device
0x00310026	Error in writing data to physical device
0x00310027	Error in cleaning the physical device
0x00310028	Error in setting events of physical device
0x00310029	Error in setting RTS setting of physical device
0x0031002A	Error in reading events of physical device
0x0031002B	Error in reading data from physical device
0x00310030	Transmission/reception frames exhausted
0x00310031	Transmission timeout
0x00310032	Reception timeout
0x00310033	Reception overrun
0x00310034	Error in receiving physical device frame
0x00310035	Physical device parity error during reception
0x00310036	Physical device reception overrun
0x00310037	Reading from physical device inconsistent
0x003100F0	Internal error - Invalid message class
0x003100F1	Internal error - Invalid message command

Contacts:**PRIMA ELECTRO S.p.A.**

Strada Carignano, 48/2 - Moncalieri (TO)

ITALY

Tel. +39 0119899800

Web: www.primaelectro.come-mail: sales@primaelectro.com

Copyright © 2013 by PRIMA ELECTRO S.p.A.