

# XPS Unified

*Universal High-Performance  
Motion Controller/Driver*



**Newport®**

Experience | Solutions

**Programmer's Manual**

**V1.0.x**

Precision Motion – **Guaranteed™**

# Preface

## **Confidentiality & Proprietary Rights**

---

### **Reservation of Title**

The Newport Programs and all materials furnished or produced in connection with them ("Related Materials") contain trade secrets of Newport and are for use only in the manner expressly permitted. Newport claims and reserves all rights and benefits afforded under law in the Programs provided by Newport Corporation.

Newport shall retain full ownership of Intellectual Property Rights in and to all development, process, align or assembly technologies developed and other derivative work that may be developed by Newport. Customer shall not challenge, or cause any third party to challenge, the rights of Newport.

### **Preservation of Secrecy and Confidentiality and Restrictions to Access**

Customer shall protect the Newport Programs and Related Materials as trade secrets of Newport, and shall devote its best efforts to ensure that all its personnel protect the Newport Programs as trade secrets of Newport Corporation. Customer shall not at any time disclose Newport's trade secrets to any other person, firm, organization, or employee that does not need (consistent with Customer's right of use hereunder) to obtain access to the Newport Programs and Related Materials. These restrictions shall not apply to information (1) generally known to the public or obtainable from public sources; (2) readily apparent from the keyboard operations, visual display, or output reports of the Programs; (3) previously in the possession of Customer or subsequently developed or acquired without reliance on the Newport Programs; or (4) approved by Newport for release without restriction.

©2016 Newport Corporation  
1791 Deere Ave.  
Irvine, CA 92606, USA  
(949) 863-3144

# Table of Contents

Preface .....	ii
<b>1. Note .....</b>	<b>1</b>
<b>2. TCP/IP Communication.....</b>	<b>2</b>
<b>3. XPS Standard Firmware Architecture (Base Version).....</b>	<b>3</b>
3.1 Group Definition .....	3
3.1.1 Object Structure .....	3
3.2 Positioner Definition .....	4
3.2.1 Object Structure .....	4
3.2.2 Definition of the Positions Available for Each Positioner .....	5
3.3 SingleAxis Group.....	6
3.3.1 State Diagram.....	7
3.4 Spindle Group .....	8
3.4.1 State Diagram.....	9
3.5 XY Group.....	10
3.5.1 State Diagram.....	11
3.6 XYZ Group .....	12
3.6.1 State Diagram.....	13
3.7 MultipleAxes Group.....	14
3.7.1 State Diagram.....	15
3.8 Analog and Digital I/O .....	16
3.8.1 GPIO Name List .....	16
3.8.1.1 ISA Hardware .....	16
3.8.1.2 PCI Hardware .....	17
<b>4. XPS Extended Firmware Architecture (Contact Newport) .....</b>	<b>18</b>
4.1 SingleAxisWithClamping Group .....	18
4.1.1 State Diagram.....	19
4.1.2 Group Clamping Sequence .....	20
4.1.2.1 Clamping state diagram .....	20
4.2 SingleAxisTheta Group.....	21
4.2.1 State Diagram.....	22
4.2.2 Group Clamping Sequence .....	23
4.2.2.1 Clamping state diagram .....	23
4.3 TZ Group.....	24
4.3.1 State Diagram.....	25

4.4	User External Module Programming .....	26
4.5	ZYGO Interferometer.....	27
4.5.1	ZMI Measurement System.....	27
4.5.2	ZYGO P2 Interface Registers .....	27
4.5.3	Status, Error, and High Nibble P2 Interface Register .....	28
4.5.4	ZYGO Error Code Table.....	29
4.5.5	PEG Control Register .....	30
4.5.6	ZYGO Axis Error Status List.....	30
4.5.7	ZYGO Axis Status List.....	31
<b>5.</b>	<b>XPS .NET Software Drivers .....</b>	<b>32</b>
5.1	How to install .NET Drivers for XPS Controller .....	32
5.1.1	Requirements .....	32
5.1.2	Installing the 32 bit (x86) Windows platform.....	33
5.1.3	Installing the 64 bit (x64) Windows platform.....	34
5.2	How to Use XPS .NET Assembly from Visual Studio C#? .....	35
5.2.1	Add Reference to .NET Assembly.....	35
5.2.2	C# Code Sources.....	36
5.3	How to use XPS .NET Assembly from LabVIEW?.....	37
5.3.1	Add Reference to .NET Assembly.....	37
5.3.2	LabVIEW Code Sources .....	38
5.4	How to use XPS .NET assembly from IronPython? .....	39
5.4.1	Add Reference to .NET Assembly.....	39
5.4.2	IronPython Code Source .....	39
5.5	How to Use XPS .NET Assembly from Matlab? .....	41
5.5.1	Add Reference to .NET Assembly.....	41
5.5.2	Matlab Code Source .....	41
<b>6.</b>	<b>XPS Functions Description .....</b>	<b>42</b>
6.1	Input Tests Common to all XPS Functions .....	42
6.2	XPS Functions Lists .....	43
6.2.1	Standard Functions.....	43
6.2.1.1	CleanCoreDumpFolder .....	43
6.2.1.2	CleanTmpFolder .....	44
6.2.1.3	CloseAllOtherSockets .....	45
6.2.1.4	ControllerMotionKernelTimeLoadGet .....	46
6.2.1.5	ControllerRTTTimeGet .....	47
6.2.1.6	ControllerSlaveStatusGet .....	48
6.2.1.7	ControllerSlaveStatusStringGet .....	49
6.2.1.8	ControllerStatusGet .....	50
6.2.1.9	ControllerStatusRead .....	51
6.2.1.10	ControllerStatusStringGet .....	52
6.2.1.11	ControllerSynchronizeCorrectorISR .....	53
6.2.1.12	DoubleGlobalArrayGet .....	54
6.2.1.13	DoubleGlobalArraySet .....	55
6.2.1.14	Elapsed Time Get .....	56

6.2.1.15	ErrorStringGet .....	57
6.2.1.16	EventExtendedAllGet .....	58
6.2.1.17	EventExtendedConfigurationActionGet .....	59
6.2.1.18	EventExtendedConfigurationActionSet .....	60
6.2.1.19	EventExtendedConfigurationTriggerGet .....	62
6.2.1.20	EventExtendedConfigurationTriggerSet .....	63
6.2.1.21	EventExtendedGet .....	65
6.2.1.22	EventExtendedRemove .....	66
6.2.1.23	EventExtendedStart .....	67
6.2.1.24	EventExtendedWait .....	69
6.2.1.25	ExternalModuleFirmwareVersionGet .....	71
6.2.1.26	ExternalModuleScanFuncTimeDurationsGet .....	72
6.2.1.27	ExternalModuleSocketFree .....	73
6.2.1.28	ExternalModuleSocketReserve .....	74
6.2.1.29	FileGatheringRename .....	75
6.2.1.30	FileScriptHistoryRename .....	76
6.2.1.31	FirmwareBuildVersionNumberGet .....	77
6.2.1.32	FirmwareVersionGet .....	78
6.2.1.33	GatheringConfigurationGet .....	79
6.2.1.34	GatheringConfigurationSet .....	80
6.2.1.35	GatheringCurrentNumberGet .....	81
6.2.1.36	GatheringDataAcquire .....	82
6.2.1.37	GatheringDataGet .....	83
6.2.1.38	GatheringDataMultipleLinesGet .....	84
6.2.1.39	GatheringExternalConfigurationGet .....	86
6.2.1.40	GatheringExternalConfigurationSet .....	87
6.2.1.41	GatheringExternalCurrentNumberGet .....	88
6.2.1.42	GatheringExternalDataGet .....	89
6.2.1.43	GatheringExternalStopAndSave .....	90
6.2.1.44	GatheringReset .....	91
6.2.1.45	GatheringRun .....	92
6.2.1.46	GatheringRunAppend .....	93
6.2.1.47	GatheringStop .....	94
6.2.1.48	GatheringStopAndSave .....	95
6.2.1.49	GetLibraryVersion .....	96
6.2.1.50	GlobalArrayGet .....	97
6.2.1.51	GlobalArraySet .....	98
6.2.1.52	GPIOAnalogGainGet .....	99
6.2.1.53	GPIOAnalogGainSet .....	100
6.2.1.54	GPIOAnalogGet .....	101
6.2.1.55	GPIOAnalogRangeConfigurationGet .....	102
6.2.1.56	GPIOAnalogRangeConfigurationSet .....	103
6.2.1.57	GPIOAnalogSet .....	104
6.2.1.58	GPIODigitalGet .....	105
6.2.1.59	GPIODigitalSet .....	106
6.2.1.60	GPIODigitalPulseWidthGet .....	107
6.2.1.61	GPIODigitalPulseWidthSet .....	108
6.2.1.62	GroupAccelerationCurrentGet .....	109
6.2.1.63	GroupAccelerationSetpointGet .....	110
6.2.1.64	GroupAnalogTrackingModeDisable .....	111
6.2.1.65	GroupAnalogTrackingModeEnable .....	112

6.2.1.66	GroupBrakeStateGet.....	114
6.2.1.67	GroupBrakeSet .....	115
6.2.1.68	GroupCorrectorOutputGet .....	116
6.2.1.69	GroupCurrentFollowingErrorGet .....	117
6.2.1.70	GroupGantryModeGet .....	118
6.2.1.71	GroupGantryModeSet.....	119
6.2.1.72	GroupHomeSearch .....	120
6.2.1.73	GroupHomeSearchAndRelativeMove .....	122
6.2.1.74	GroupInitialize .....	124
6.2.1.75	GroupInitializeNoEncoderReset.....	126
6.2.1.76	GroupInitializeWithEncoderCalibration.....	128
6.2.1.77	GroupInterlockDisable.....	130
6.2.1.78	GroupInterlockEnable.....	131
6.2.1.79	GroupJogCurrentGet.....	132
6.2.1.80	GroupJogModeDisable .....	133
6.2.1.81	GroupJogModeEnable .....	134
6.2.1.82	GroupJogParametersGet .....	136
6.2.1.83	GroupJogParametersSet.....	137
6.2.1.84	GroupKill.....	139
6.2.1.85	GroupMotionDisable .....	140
6.2.1.86	GroupMotionEnable .....	141
6.2.1.87	GroupMotionStatusGet .....	142
6.2.1.88	GroupMoveAbort .....	143
6.2.1.89	GroupMoveAbortFast.....	145
6.2.1.90	GroupMoveAbsolute .....	147
6.2.1.91	GroupMoveEndWait.....	149
6.2.1.92	GroupMoveRelative .....	150
6.2.1.93	GroupPositionCurrentGet .....	152
6.2.1.94	GroupPositionSetpointGet .....	153
6.2.1.95	GroupPositionTargetGet.....	154
6.2.1.96	GroupReferencingActionExecute .....	155
6.2.1.97	GroupReferencingStart .....	157
6.2.1.98	GroupReferencingStop .....	158
6.2.1.99	GroupSpinCurrentGet .....	159
6.2.1.100	GroupSpinModeStop .....	160
6.2.1.101	GroupSpinParametersGet .....	161
6.2.1.102	GroupSpinParametersSet.....	162
6.2.1.103	GroupStatusGet.....	164
6.2.1.104	GroupStatusStringGet.....	165
6.2.1.105	GroupVelocityCurrentGet .....	166
6.2.1.106	GroupVelocitySetpointGet .....	167
6.2.1.107	HardwareDateAndTimeGet .....	168
6.2.1.108	HardwareDateAndTimeSet.....	169
6.2.1.109	HardwareDriverAndStageGet.....	170
6.2.1.110	InstallerVersionGet.....	171
6.2.1.111	INTServitudesCommandGet .....	172
6.2.1.112	INTServitudesStatusGet .....	173
6.2.1.113	KillAll.....	174
6.2.1.114	Login.....	175
6.2.1.115	MultipleAxesPTExecution.....	176
6.2.1.116	MultipleAxesPTLoadToMemory .....	178

6.2.1.117	MultipleAxesPTParametersGet .....	180
6.2.1.118	MultipleAxesPTPulseOutputGet .....	181
6.2.1.119	MultipleAxesPTPulseOutputSet .....	183
6.2.1.120	MultipleAxesPTResetInMemory .....	185
6.2.1.121	MultipleAxesPTVerification .....	186
6.2.1.122	MultipleAxesPTVerificationResultGet .....	188
6.2.1.123	MultipleAxesPVTExecution .....	189
6.2.1.124	MultipleAxesPVTLoadToMemory .....	191
6.2.1.125	MultipleAxesPVTParametersGet .....	193
6.2.1.126	MultipleAxesPVTpulseOutputGet .....	194
6.2.1.127	MultipleAxesPVTpulseOutputSet .....	196
6.2.1.128	MultipleAxesPVTResetInMemory .....	198
6.2.1.129	MultipleAxesPVTVerification .....	199
6.2.1.130	MultipleAxesPVTVerificationResultGet .....	201
6.2.1.131	OpenConnection .....	202
6.2.1.132	PositionerAccelerationAutoScaling .....	203
6.2.1.133	PositionerAnalogTrackingPositionParametersGet .....	205
6.2.1.134	PositionerAnalogTrackingPositionParametersSet .....	206
6.2.1.135	PositionerAnalogTrackingVelocityParametersGet .....	208
6.2.1.136	PositionerAnalogTrackingVelocityParametersSet .....	210
6.2.1.137	PositionerBacklashDisable .....	212
6.2.1.138	PositionerBacklashEnable .....	213
6.2.1.139	PositionerBacklashGet .....	215
6.2.1.140	PositionerBacklashSet .....	216
6.2.1.141	PositionerCompensatedFastPCOAbort .....	217
6.2.1.142	PositionerCompensatedFastPCOCurrentStatusGet .....	218
6.2.1.143	PositionerCompensatedFastPCOEnable .....	219
6.2.1.144	PositionerCompensatedFastPCOFromFile .....	220
6.2.1.145	PositionerCompensatedFastPCOLoadToMemory .....	222
6.2.1.146	PositionerCompensatedFastPCOMemoryReset .....	223
6.2.1.147	PositionerCompensatedFastPCOPrepare .....	224
6.2.1.148	PositionerCompensatedFastPCOPulseParametersGet .....	226
6.2.1.149	PositionerCompensatedFastPCOPulseParametersSet .....	227
6.2.1.150	PositionerCompensatedFastPCOSet .....	228
6.2.1.151	PositionerCompensatedPCOAbort .....	230
6.2.1.152	PositionerCompensatedPCOCurrentStatusGet .....	232
6.2.1.153	PositionerCompensatedPCOEnable .....	233
6.2.1.154	PositionerCompensatedPCOFromFile .....	235
6.2.1.155	PositionerCompensatedPCOLoadToMemory .....	237
6.2.1.156	PositionerCompensatedPCOMemoryReset .....	239
6.2.1.157	PositionerCompensatedPCOPrepare .....	240
6.2.1.158	PositionerCompensatedPCOSet .....	242
6.2.1.159	PositionerCompensationDisturbanceDisable .....	244
6.2.1.160	PositionerCompensationDisturbanceEnable .....	245
6.2.1.161	PositionerCompensationDisturbanceFileLoad .....	246
6.2.1.162	PositionerCompensationDisturbanceStatusGet .....	247
6.2.1.163	PositionerCompensationDualLoopNotchFilterGet .....	248
6.2.1.164	PositionerCompensationDualLoopNotchFilterSet .....	250
6.2.1.165	PositionerCompensationDualLoopPhaseCorrectionFilterGet .....	252
6.2.1.166	PositionerCompensationDualLoopPhaseCorrectionFilterSet .....	254
6.2.1.167	PositionerCompensationEncoderNotchFilterGet .....	256

6.2.1.168 PositionerCompensationEncoderNotchFilterSet .....	258
6.2.1.169 PositionerCompensationFrequencyNotchsGet .....	260
6.2.1.170 PositionerCompensationFrequencyNotchsSet .....	262
6.2.1.171 PositionerCompensationLowPassTwoFilterGet .....	264
6.2.1.172 PositionerCompensationLowPassTwoFilterSet .....	265
6.2.1.173 PositionerCompensationNotchFilterGet .....	266
6.2.1.174 PositionerCompensationNotchFilterSet .....	267
6.2.1.175 PositionerCompensationNotchModeFiltersGet .....	269
6.2.1.176 PositionerCompensationNotchModeFiltersSet .....	271
6.2.1.177 PositionerCompensationPhaseCorrectionFilterGet .....	273
6.2.1.178 PositionerCompensationPhaseCorrectionFilterSet .....	275
6.2.1.179 PositionerCompensationPhaseCorrectionFiltersGet .....	277
6.2.1.180 PositionerCompensationPhaseCorrectionFiltersSet .....	279
6.2.1.181 PositionerCompensationPositionFilterGet .....	281
6.2.1.182 PositionerCompensationPositionFilterSet .....	282
6.2.1.183 PositionerCompensationPostExcitationFrequencyNotchFilterGet .....	283
6.2.1.184 PositionerCompensationPostExcitationFrequencyNotchFilterSet .....	284
6.2.1.185 PositionerCompensationPostExcitationLowPassFilterGet .....	286
6.2.1.186 PositionerCompensationPostExcitationLowPassFilterSet .....	287
6.2.1.187 PositionerCompensationPostExcitationNotchModeFilterGet .....	288
6.2.1.188 PositionerCompensationPostExcitationNotchModeFilterSet .....	290
6.2.1.189 PositionerCompensationPostExcitationPhaseCorrectionFilterGet .....	292
6.2.1.190 PositionerCompensationPostExcitationPhaseCorrectionFilterSet .....	294
6.2.1.191 PositionerCompensationPreFeedForwardFrequencyNotchFilterGet .....	296
6.2.1.192 PositionerCompensationPreFeedForwardFrequencyNotchFilterSet .....	297
6.2.1.193 PositionerCompensationPreFeedForwardPhaseCorrectionFilterGet .....	299
6.2.1.194 PositionerCompensationPreFeedForwardPhaseCorrectionFilterSet .....	301
6.2.1.195 PositionerCompensationPreFeedForwardSpatialNotchFilterGet .....	303
6.2.1.196 PositionerCompensationPreFeedForwardSpatialNotchFilterSet .....	304
6.2.1.197 PositionerCompensationSpatialPeriodicNotchsGet .....	306
6.2.1.198 PositionerCompensationSpatialPeriodicNotchsSet .....	308
6.2.1.199 PositionerCorrectorAutoTuning .....	310
6.2.1.200 PositionerCorrectorDamperFilterGet .....	312
6.2.1.201 PositionerCorrectorDamperFilterSet .....	313
6.2.1.202 PositionerCorrectorDualGet .....	315
6.2.1.203 PositionerCorrectorDualSet .....	317
6.2.1.204 PositionerCorrectorExcitationSignalGainGet .....	319
6.2.1.205 PositionerCorrectorExcitationSignalGainSet .....	320
6.2.1.206 PositionerCorrectorNotchFiltersGet .....	321
6.2.1.207 PositionerCorrectorNotchFiltersSet .....	323
6.2.1.208 PositionerCorrectorPIDAccelerationFilterGet .....	325
6.2.1.209 PositionerCorrectorPIDAccelerationFilterSet .....	326
6.2.1.210 PositionerCorrectorPIDBaseGet .....	327
6.2.1.211 PositionerCorrectorPIDBaseSet .....	329
6.2.1.212 PositionerCorrectorPIDDualFFVoltageGet .....	331
6.2.1.213 PositionerCorrectorPIDDualFFVoltageSet .....	333
6.2.1.214 PositionerCorrectorPIDFFAccelerationGet .....	336
6.2.1.215 PositionerCorrectorPIDFFAccelerationSet .....	338
6.2.1.216 PositionerCorrectorPIDFFVelocityGet .....	341
6.2.1.217 PositionerCorrectorPIDFFVelocitySet .....	343
6.2.1.218 PositionerCorrectorPIPositionGet .....	345

6.2.1.219 PositionerCorrectorPIPositionSet .....	346
6.2.1.220 PositionerCorrectorPlantFeedForwardDelayGet .....	348
6.2.1.221 PositionerCorrectorPlantFeedForwardDelaySet .....	349
6.2.1.222 PositionerCorrectorPostFFGet .....	350
6.2.1.223 PositionerCorrectorPostFFSet .....	351
6.2.1.224 PositionerCorrectorTypeGet .....	352
6.2.1.225 PositionerCurrentVelocityAccelerationFiltersGet .....	353
6.2.1.226 PositionerCurrentVelocityAccelerationFiltersSet .....	354
6.2.1.227 PositionerDriverFiltersGet .....	355
6.2.1.228 PositionerDriverFiltersSet .....	356
6.2.1.229 PositionerDriverPositionOffsetsGet .....	358
6.2.1.230 PositionerDriverStatusGet .....	359
6.2.1.231 PositionerDriverStatusStringGet .....	360
6.2.1.232 PositionerEncoderAmplitudeValuesGet .....	361
6.2.1.233 PositionerEncoderCalibrationParametersGet .....	362
6.2.1.234 PositionersEncoderIndexDifferenceGet .....	364
6.2.1.235 PositionerErrorGet .....	365
6.2.1.236 PositionerErrorRead .....	366
6.2.1.237 PositionerErrorStringGet .....	367
6.2.1.238 PositionerExcitationSignalGet .....	368
6.2.1.239 PositionerExcitationSignalSet .....	369
6.2.1.240 PositionerFeedforwardAccDisable .....	372
6.2.1.241 PositionerFeedforwardAccEnable .....	373
6.2.1.242 PositionerFeedforwardAccGet .....	374
6.2.1.243 PositionerFeedforwardAccSet .....	375
6.2.1.244 PositionerFeedforwardPositionDisable .....	376
6.2.1.245 PositionerFeedforwardPositionEnable .....	377
6.2.1.246 PositionerFeedforwardPositionGet .....	378
6.2.1.247 PositionerFeedforwardPositionSet .....	379
6.2.1.248 PositionerGantryEndReferencingPositionGet .....	380
6.2.1.249 PositionerHardInterpolatorFactorGet .....	381
6.2.1.250 PositionerHardInterpolatorFactorSet .....	382
6.2.1.251 PositionerHardInterpolatorPositionGet .....	384
6.2.1.252 PositionerHardwareStatusGet .....	385
6.2.1.253 PositionerHardwareStatusStringGet .....	386
6.2.1.254 PositionerJogMaximumVelocityAndAccelerationGet .....	387
6.2.1.255 PositionerMagneticTrackPositionAtHomeGet .....	388
6.2.1.256 PositionerMaximumVelocityAndAccelerationGet .....	389
6.2.1.257 PositionerMotionDoneGet .....	390
6.2.1.258 PositionerMotionDoneSet .....	391
6.2.1.259 PositionerPositionCompareAquadBAlwaysEnable .....	392
6.2.1.260 PositionerPositionCompareAquadBPrescalerGet .....	393
6.2.1.261 PositionerPositionCompareAquadBPrescalerSet .....	394
6.2.1.262 PositionerPositionCompareAquadBWindowedGet .....	395
6.2.1.263 PositionerPositionCompareAquadBWindowedSet .....	396
6.2.1.264 PositionerPositionCompareDisable .....	398
6.2.1.265 PositionerPositionCompareEnable .....	399
6.2.1.266 PositionerPositionCompareGet .....	400
6.2.1.267 PositionerPositionComparePulseParametersGet .....	401
6.2.1.268 PositionerPositionComparePulseParametersSet .....	402
6.2.1.269 PositionerPositionCompareScanAccelerationLimitGet .....	404

6.2.1.270 PositionerPositionCompareScanAccelerationLimitSet.....	405
6.2.1.271 PositionerPositionCompareSet .....	406
6.2.1.272 PositionerPreCorrectorExcitationSignalGet .....	408
6.2.1.273 PositionerPreCorrectorExcitationSignalSet.....	409
6.2.1.274 PositionerRawEncoderPositionGet.....	411
6.2.1.275 PositionerSGammaExactVelocityAjustedDisplacementGet.....	412
6.2.1.276 PositionerSGammaParametersGet.....	413
6.2.1.277 PositionerSGammaParametersSet.....	414
6.2.1.278 PositionerSGammaPreviousMotionTimesGet .....	415
6.2.1.279 PositionerSGammaVelocityAndAccelerationSet .....	416
6.2.1.280 PositionerStageParameterGet .....	417
6.2.1.281 PositionerStageParameterSet .....	418
6.2.1.282 PositionerTimeFlasherDisable .....	419
6.2.1.283 PositionerTimeFlasherEnable .....	420
6.2.1.284 PositionerTimeFlasherGet .....	421
6.2.1.285 PositionerTimeFlasherSet .....	422
6.2.1.286 PositionerUserTravelLimitsGet .....	424
6.2.1.287 PositionerUserTravelLimitsSet.....	425
6.2.1.288 PositionerWarningFollowingErrorGet.....	426
6.2.1.289 PositionerWarningFollowingErrorSet .....	427
6.2.1.290 Reboot.....	428
6.2.1.291 RestartApplication .....	429
6.2.1.292 RunPidin .....	430
6.2.1.293 SingleAxisSlaveModeDisable .....	431
6.2.1.294 SingleAxisSlaveModeEnable .....	432
6.2.1.295 SingleAxisSlaveParametersGet .....	433
6.2.1.296 SingleAxisSlaveParametersSet.....	434
6.2.1.297 SingleAxisThetaClampDisable.....	436
6.2.1.298 SingleAxisThetaClampEnable .....	437
6.2.1.299 SingleAxisThetaFeedforwardJerkParametersGet .....	438
6.2.1.300 SingleAxisThetaFeedforwardJerkParametersSet .....	439
6.2.1.301 SingleAxisThetaFeedforwardParametersGet .....	440
6.2.1.302 SingleAxisThetaFeedforwardParametersSet .....	441
6.2.1.303 SingleAxisThetaSlaveModeDisable .....	442
6.2.1.304 SingleAxisThetaSlaveModeEnable .....	443
6.2.1.305 SingleAxisThetaSlaveParametersGet .....	444
6.2.1.306 SingleAxisThetaSlaveParametersSet .....	445
6.2.1.307 SpindleSlaveModeDisable .....	447
6.2.1.308 SpindleSlaveModeEnable .....	448
6.2.1.309 SpindleSlaveParametersGet.....	449
6.2.1.310 SpindleSlaveParametersSet .....	450
6.2.1.311 TCLScriptExecute .....	452
6.2.1.312 TCLScriptExecuteAndWait.....	453
6.2.1.313 TCLScriptExecuteWithPriority .....	455
6.2.1.314 TCLScriptKill .....	457
6.2.1.315 TCLScriptKillAll .....	458
6.2.1.316 TCLScriptRunningListGet.....	459
6.2.1.317 TCP_CloseSocket .....	460
6.2.1.318 TCP_ConnectToServer.....	461
6.2.1.319 TCP_GetError.....	462
6.2.1.320 TCP_SetTimeout .....	463

6.2.1.321	TimerGet.....	464
6.2.1.322	TimerSet .....	465
6.2.1.323	TZFocusModeDisable .....	466
6.2.1.324	TZFocusModeEnable .....	467
6.2.1.325	TZMotorDecouplingMatrixGet .....	468
6.2.1.326	TZMotorDecouplingMatrixSet.....	470
6.2.1.327	TZPTExecution.....	472
6.2.1.328	TZPTLoadToMemory .....	474
6.2.1.329	TZPTParametersGet .....	476
6.2.1.330	TZPTPulseOutputGet .....	477
6.2.1.331	TZPTPulseOutputSet .....	479
6.2.1.332	TZPTResetInMemory.....	481
6.2.1.333	TZPTVerification .....	482
6.2.1.334	TZPTVerificationResultGet.....	484
6.2.1.335	TZPVTExecution.....	486
6.2.1.336	TZPVTLoadToMemory.....	488
6.2.1.337	TZPVTParametersGet .....	490
6.2.1.338	TZPVTPulseOutputGet .....	491
6.2.1.339	TZPVTPulseOutputSet .....	493
6.2.1.340	TZPVTResetInMemory .....	495
6.2.1.341	TZPVTVerification.....	496
6.2.1.342	TZPVTVerificationResultGet.....	498
6.2.1.343	TZTrackingCutOffFrequencyGet .....	499
6.2.1.344	TZTrackingCutOffFrequencySet .....	500
6.2.1.345	TZTrackingUserMaximumZZZTargetDifferenceGet.....	501
6.2.1.346	TZTrackingUserMaximumZZZTargetDifferenceSet .....	502
6.2.1.347	XYGroupPositionCorrectedProfilerGet.....	503
6.2.1.348	XYGroupPositionPCORawEncoderGet .....	505
6.2.1.349	XYLineArcExecution .....	506
6.2.1.350	XYLineArcParametersGet.....	508
6.2.1.351	XYLineArcPulseOutputGet .....	509
6.2.1.352	XYLineArcPulseOutputSet .....	511
6.2.1.353	XYLineArcVerification .....	513
6.2.1.354	XYLineArcVerificationResultGet .....	515
6.2.1.355	XYMappingGet .....	516
6.2.1.356	XYMappingSet .....	518
6.2.1.357	XYPTExecution.....	520
6.2.1.358	XYPTLoadToMemory.....	522
6.2.1.359	XYPTParametersGet .....	524
6.2.1.360	XYPTPulseOutputGet .....	525
6.2.1.361	XYPTPulseOutputSet .....	527
6.2.1.362	XYPTResetInMemory .....	529
6.2.1.363	XYPTVerification.....	530
6.2.1.364	XYPTVerificationResultGet.....	532
6.2.1.365	XYPVTExecution.....	534
6.2.1.366	XYPVTLoadToMemory.....	536
6.2.1.367	XYPVTParametersGet .....	538
6.2.1.368	XYPVTPulseOutputGet .....	539
6.2.1.369	XYPVTPulseOutputSet .....	541
6.2.1.370	XYPVTResetInMemory .....	543
6.2.1.371	XYPVTVerification.....	544

6.2.1.372	XYPVTVerificationResultGet.....	546
6.2.1.373	XYZGroupPositionCorrectedProfilerGet .....	548
6.2.1.374	XYZGroupPositionPCORawEncoderGet.....	550
6.2.1.375	XYZSplineExecution.....	552
6.2.1.376	XYZSplineParametersGet .....	554
6.2.1.377	XYZSplinePulseOutputGet .....	555
6.2.1.378	XYZSplinePulseOutputSet .....	557
6.2.1.379	XYZSplineVerification.....	559
6.2.1.380	XYZSplineVerificationResultGet.....	561
6.2.2	Extended Functions.....	562
6.2.2.1	AbortMove.....	562
6.2.2.2	EndJog .....	563
6.2.2.3	GetAccParams .....	564
6.2.2.4	GetBrakeState .....	565
6.2.2.5	GetCurrentPosition .....	566
6.2.2.6	GetGantryMode .....	567
6.2.2.7	GetJogAcceleration.....	568
6.2.2.8	GetJogVelocity .....	569
6.2.2.9	GetPistonState.....	570
6.2.2.10	GetVarX.....	572
6.2.2.11	GetVarXSecondary .....	573
6.2.2.12	GetVarY.....	574
6.2.2.13	GetVelParams .....	575
6.2.2.14	GetVerCommand .....	576
6.2.2.15	GetXVelParams .....	577
6.2.2.16	GetYVelParams .....	578
6.2.2.17	GetZone .....	579
6.2.2.18	InitializeAndHomeX.....	580
6.2.2.19	InitializeAndHomeXY .....	582
6.2.2.20	InitializeAndHomeY .....	584
6.2.2.21	MoveAbsolute.....	586
6.2.2.22	MoveSlice .....	588
6.2.2.23	RequestType1 .....	590
6.2.2.24	RequestType2 .....	592
6.2.2.25	RequestType3 .....	593
6.2.2.26	SetAccParams .....	594
6.2.2.27	SetBrake.....	595
6.2.2.28	SetGantryMode .....	596
6.2.2.29	SetJogAcceleration .....	598
6.2.2.30	SetJogVelocity .....	599
6.2.2.31	SetPiston .....	601
6.2.2.32	SetVarX .....	602
6.2.2.33	SetVarXSecondary .....	603
6.2.2.34	SetVarY .....	604
6.2.2.35	SetVelParams.....	605
6.2.2.36	SetXVelParams .....	606
6.2.2.37	SetYVelParams .....	607
6.2.2.38	SetZone .....	608
6.2.2.39	StartJog .....	609
6.2.2.40	WaitMotionEnd .....	610
6.2.2.41	ZygoADCDiagnosticStatusGet.....	611

6.2.2.42	ZygoAmplitudeGet .....	612
6.2.2.43	ZygoConnectToServer .....	613
6.2.2.44	ZygoDisconnectFromServer .....	614
6.2.2.45	ZygoErrorStatusGet .....	615
6.2.2.46	ZygoErrorStatusStringGet .....	616
6.2.2.47	ZygoEthernetCommunicationStatusGet .....	617
6.2.2.48	ZygoGetPEGLastCommunicationTime .....	618
6.2.2.49	ZygoGetVerInterfero .....	619
6.2.2.50	ZygoInterferometerStatusGet .....	620
6.2.2.51	ZygoPositionGet .....	621
6.2.2.52	ZygoReadLong .....	622
6.2.2.53	ZygoReadWord .....	623
6.2.2.54	ZygoRegisterGet .....	624
6.2.2.55	ZygoRegisterSet .....	625
6.2.2.56	ZygoReset .....	626
6.2.2.57	ZygoResetX .....	627
6.2.2.58	ZygoResetY .....	628
6.2.2.59	ZygoSendAndReceive .....	630
6.2.2.60	ZygoSetOffsetX .....	631
6.2.2.61	ZygoSetOffsetY .....	632
6.2.2.62	ZygoSetPEGParams .....	633
6.2.2.63	ZygoStartBoardP2 .....	635
6.2.2.64	ZygoStartInterferometer .....	637
6.2.2.65	ZygoStatusGet .....	638
6.2.2.66	ZygoStatusStringGet .....	639
6.2.2.67	ZygoWriteLong .....	640
6.2.2.68	ZygoWriteWord .....	641

---

## 7. Lists and Tables for XPS Functions..... **642**

7.1	Event Triggers List.....	642
7.2	Actions List.....	644
7.3	Gathering Data Types .....	645
7.4	External Gathering Data Types .....	647
7.5	Positioner Error List.....	648
7.6	Positioner Hardware Status List.....	649
7.7	Positioner Driver Status List .....	651
7.8	Group Status List.....	652
7.9	Error List.....	655
7.10	Controller Status List.....	659

---

## 8. Process Examples..... **660**

8.1	Management of Errors Example.....	660
8.2	Firmware Version Example .....	661
8.3	Gathering with Motion Example.....	662
8.4	External Gathering Example .....	664
8.5	Position Output Compare Example .....	666
8.6	Slave-Master Mode Example .....	668

8.7	Jogging Example.....	670
8.8	Tracking Example .....	672
8.9	Backlash .....	673
8.10	Timer Event and Global Variables .....	675
8.11	Running Several Motion Processes Simultaneously.....	677
<b>Service Form .....</b>		<b>679</b>



# XPS

## Universal High-Performance Motion Controller/Driver

### 1.

### Note

---

This unified programmer's manual describes all the functions available from the XPS family of controllers.

---

#### NOTE

**The subset of available functions depends on the version of XPS controller you are using. Refer to your controller's Web site (Terminal page) to get the exact list.**

---

---

#### NOTE

**For more details on XPS features, please refer to XPS User's Manual.**

---

## 2.

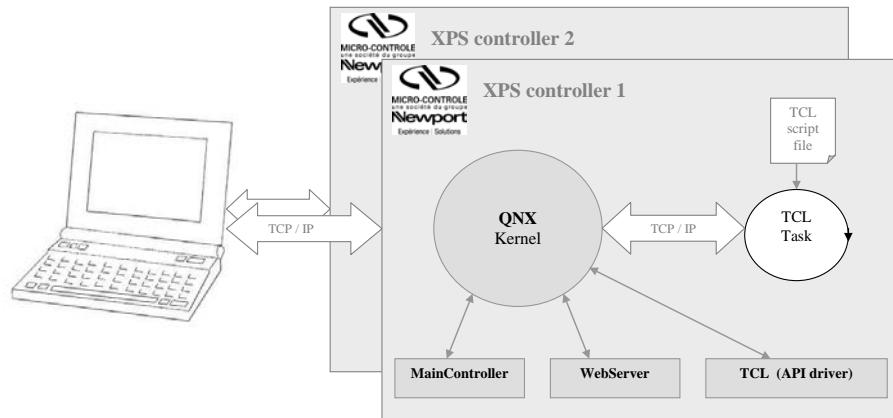
**TCP/IP Communication**

XPS is based on a 10/100 Base-T Ethernet communication link with TCP/IP protocol and uses a web site as the Graphical User Interface (GUI) to access all the software tools and an FTP server for file transfer. This makes the XPS controller independent of the user's operating system. When networked, Unix, Linux or Windows users can access the same controller from any place in the world for remote control, code development, file transfer or diagnostics. The completely object oriented approach of the XPS firmware with powerful, multi-parameter Functions (commands) is also much more consistent and intuitive to use than old-style mnemonic commands.

To connect to the XPS controller you must open a socket with the "OpenConnection()" Function. Communication through this socket is done by specifying the socket identifier (socketID) with the Function. The XPS allows up to 100 open sockets with 30 sockets communicating simultaneously.

Each Function returns a complete or error message. If successful, the return is 0 (zero). In case of an error, the returned error code (2 – digit) can be used for diagnosing the problem with the Function ErrorStringGet().

The function call is blocked until a reply is sent by the XPS, or until the timeout value is reached. For running several processes in parallel (for instance getting the position while a stage is moving), several sockets can be used in parallel. When using the XPS controller with programming languages that do not support multiple sockets, the timeout value of the function can be set to a low value (20 ms).



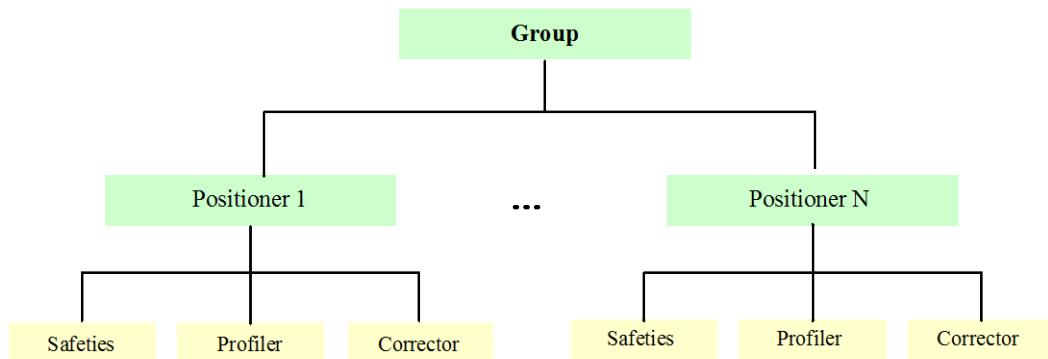
## 3. XPS Standard Firmware Architecture (Base Version)

### 3.1 Group Definition

The “Group” objects are used to define one or several “positioners” in the same motion group. The available motion groups are defined in the section [GROUPS] in the system.ini file and the group types are:

- **SingleAxis** (1 positioner)/“Gantry” **SingleAxis** (2 positioners)
- **SingleAxisWithClamping** (1 positioner)
- **SingleAxisTheta** (1 positioner)
- **Spindle** (1 positioner)
- **XY** (2 positioners)/“Gantry” **XY** (3 or 4 positioners)
- **XYZ** (3 positioners)
- **MultipleAxes**/“Gantry” **MultipleAxes** (1 to 8 positioners)
- **TZ** (3 positioners)

#### 3.1.1 Object Structure



A motion “Group” is created in relation to a **group type** (SingleAxis, SingleAxisWithClamping, SingleAxisTheta, Spindle, XY, XYZ, TZ or MultipleAxes).

A group is defined by a **group name**.

#### NOTE

**The maximum number of positioners in the same group is limited to 8.**

### 3.2 Positioner Definition

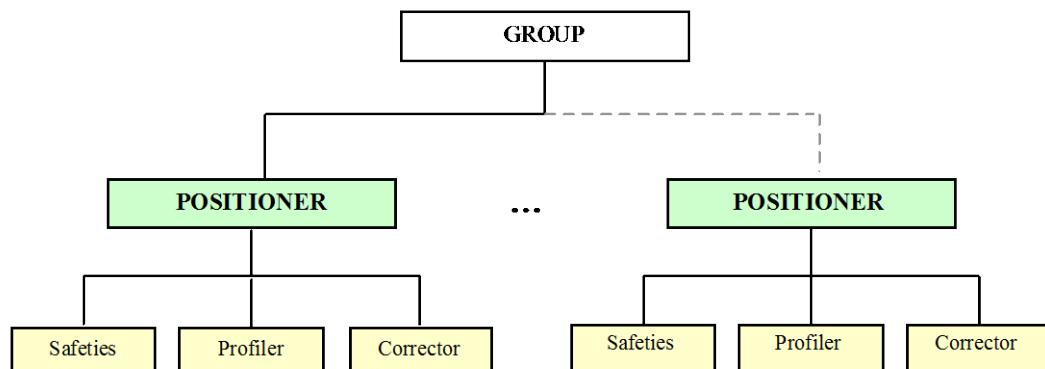
Positioner objects are used to define all motion specific configuration parameters.

The positioner includes a mapping correction:  $X = f(X)$

The positioner includes the SGamma profile.

The maximum number of positioners is limited to ControllerAxesNumber (*4, 8, 12 or 16, depending on type of the XPS controller hardware*).

#### 3.2.1 Object Structure



To use a **positioner**, it must belong to a motion group. Positioners are defined by their **full positioner name**. The full positioner name is composed of the group name and the positioner name separated by a character “.”.

*Example:*

*GroupName.PositionerName or*

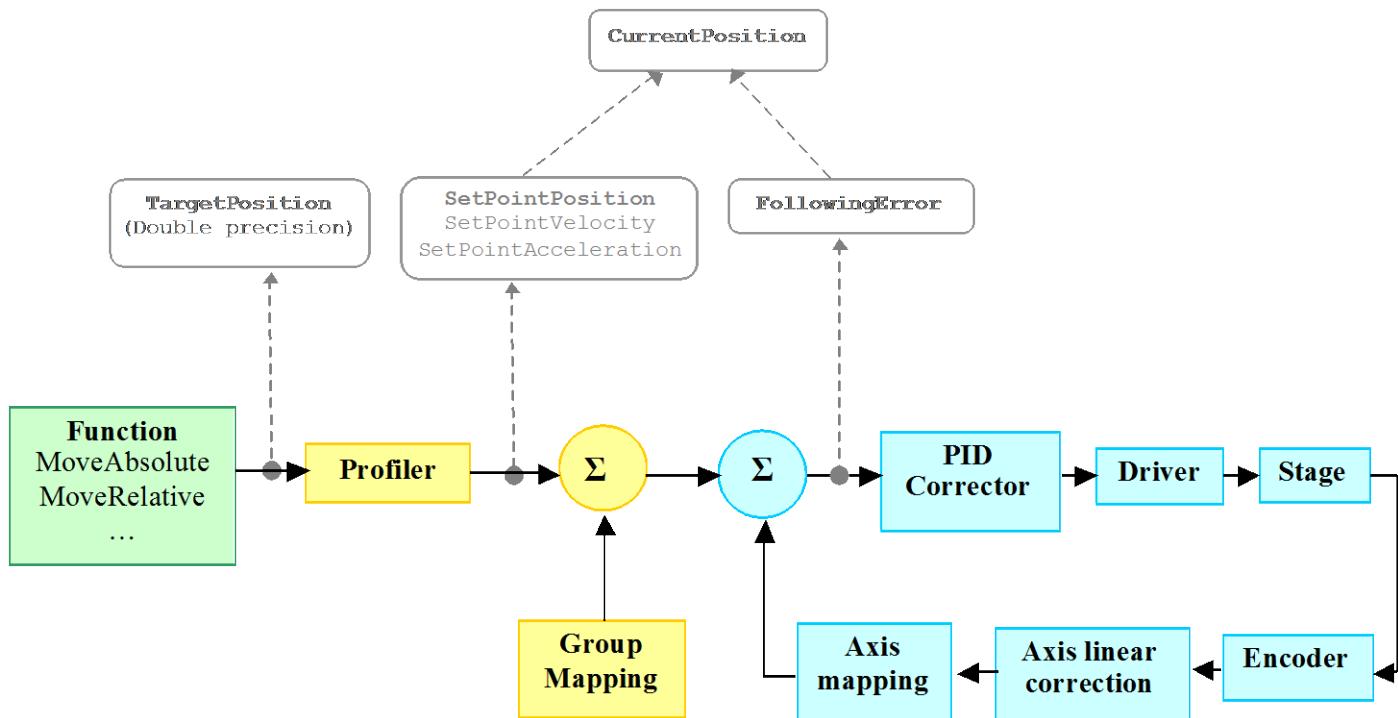
*3-Axis system.X-axis*

### 3.2.2 Definition of the Positions Available for Each Positioner

For each positioner, three different positions can be called:

1. The **SetpointPosition** is the profiler position. This is the position where the positioner should be according to the ideal theoretical motion profile.
2. The **CurrentPosition** is the encoder position of the stage after mapping corrections are applied. This is the actual position of the positioner at this moment of the query.
3. The **TargetPosition** is the final target position commanded by the user.

The difference between the SetpointPosition and the CurrentPosition is called the following error.



*For example, during a motion from the position 0 (units) to 100 (units), a query in the middle of the motion could have the following result:*

*SetpointPosition = 50*

*CurrentPosition = 49.998 (FollowingError = 50 – 49.998 = 0.002 unit)*

*TargetPosition = 100.*

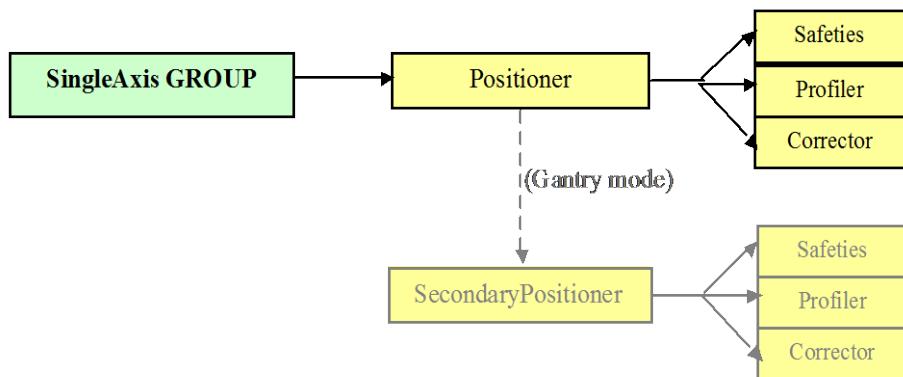
### 3.3 SingleAxis Group

The SingleAxis group is composed of one single positioner for the execution of motion commands.

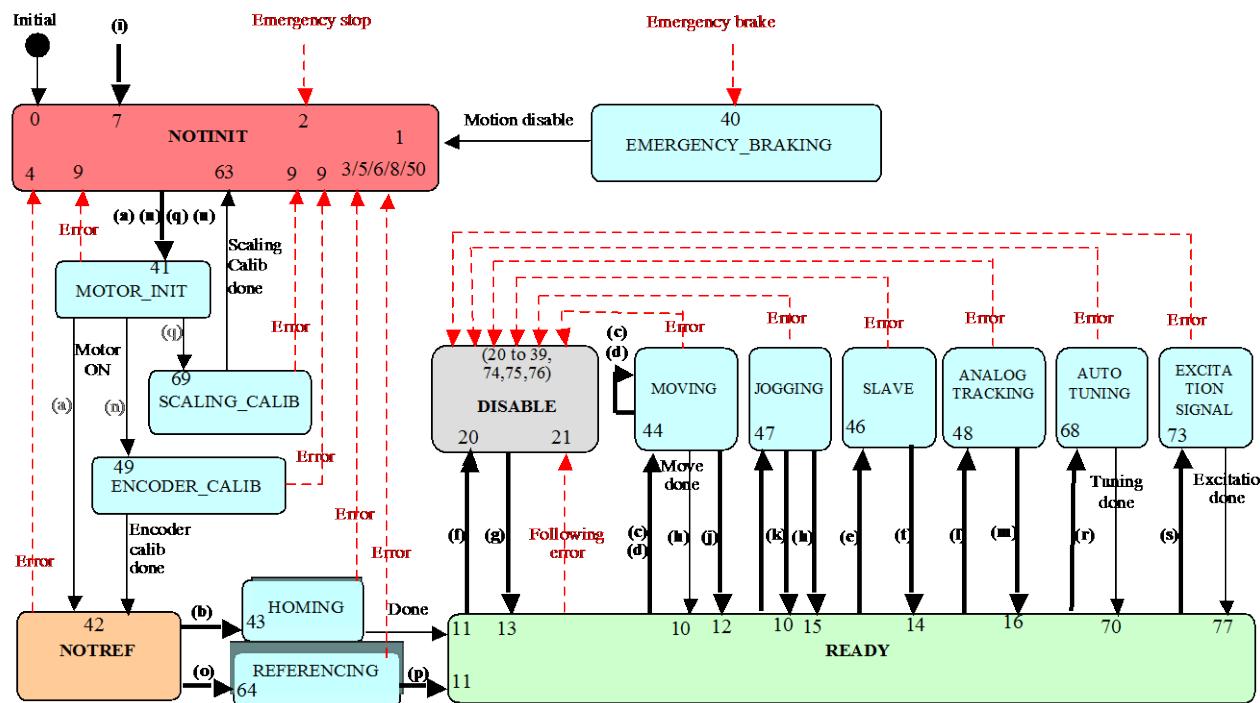
A SingleAxis group can be used in GANTRY mode (dual positioner).

The XPS controller can handle several SingleAxis objects.

There is no relation between SingleAxis objects and other objects handled by the controller.



### 3.3.1 State Diagram



#### Called functions:

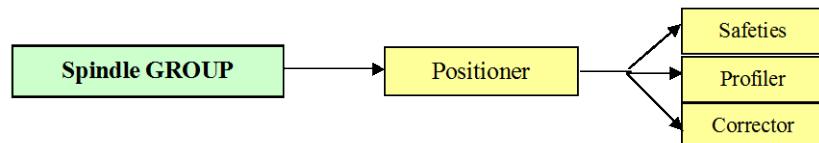
- |     |                      |     |   |
|-----|----------------------|-----|---|
| (a) | GroupInitialize      | (l) | GroupAnalogTrackingModeEnable   |
| (b) | GroupHomeSearch      | (m) | GroupAnalogTrackingModeDisable  |
| (c) | GroupMoveAbsolute    | (n) | GroupInitializeWithEncoderCalibration                                     |
| (d) | GroupMoveRelative    | (o) | GroupReferencingStart   |
| (e) | GroupSlaveModeEnable | (p) | GroupReferencingStop  |
| (f) | GroupMotionDisable   | (q) | PositionerAccelerationAutoScaling   |
| (g) | GroupMotionEnable    | (r) | PositionerCorrectorAutoTuning   |
| (h) | GroupMoveAbort       | (s) | PositionerExcitationSignalSet / PositionerPreCorrectorExcitationSignalSet |
| (i) | GroupKill or KillAll | (t) | GroupSlaveModeDisable   |
| (j) | GroupJogModeEnable   | (u) | GroupInitializeNoEncoderReset   |
| (k) | GroupJogModeDisable  |     |   |

### 3.4 Spindle Group

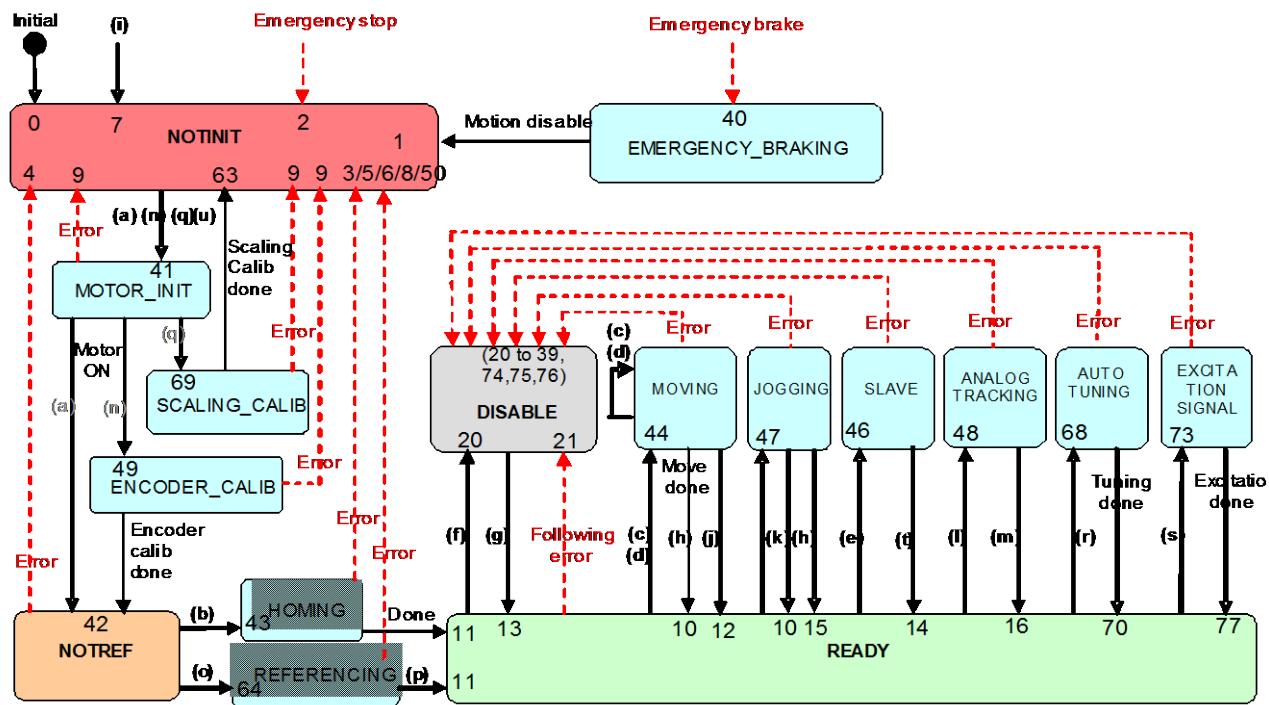
A Spindle group is very similar to the SingleAxis group. It is composed of only one positioner with one main difference, it does not handle software or hardware end of runs. Therefore, it is allowed to spin indefinitely in any direction. The SingleAxis group motion commands are still allowed (except jog, which is replaced by spin).

The controller can handle several Spindle groups.

There is no relation between Spindle groups and other objects handled by the controller.



### 3.4.1 State Diagram



#### Called functions:

- |     |                      |     |   |
|-----|----------------------|-----|---|
| (a) | GroupInitialize      | (l) | GroupAnalogTrackingModeEnable   |
| (b) | GroupHomeSearch      | (m) | GroupAnalogTrackingModeDisable  |
| (c) | GroupMoveAbsolute    | (n) | GroupInitializeWithEncoderCalibration                                     |
| (d) | GroupMoveRelative    | (o) | GroupReferencingStart   |
| (e) | GroupSlaveModeEnable | (p) | GroupReferencingStop  |
| (f) | GroupMotionDisable   | (q) | PositionerAccelerationAutoScaling   |
| (a) | GroupInitialize      | (r) | PositionerCorrectorAutoTuning   |
| (b) | GroupHomeSearch      | (s) | PositionerExcitationSignalSet / PositionerPreCorrectorExcitationSignalSet |
| (c) | GroupMoveAbsolute    | (t) | GroupSlaveModeDisable   |
| (d) | GroupMoveRelative    | (u) | GroupInitializeNoEncoderReset   |
| (e) | GroupSlaveModeEnable |     |   |

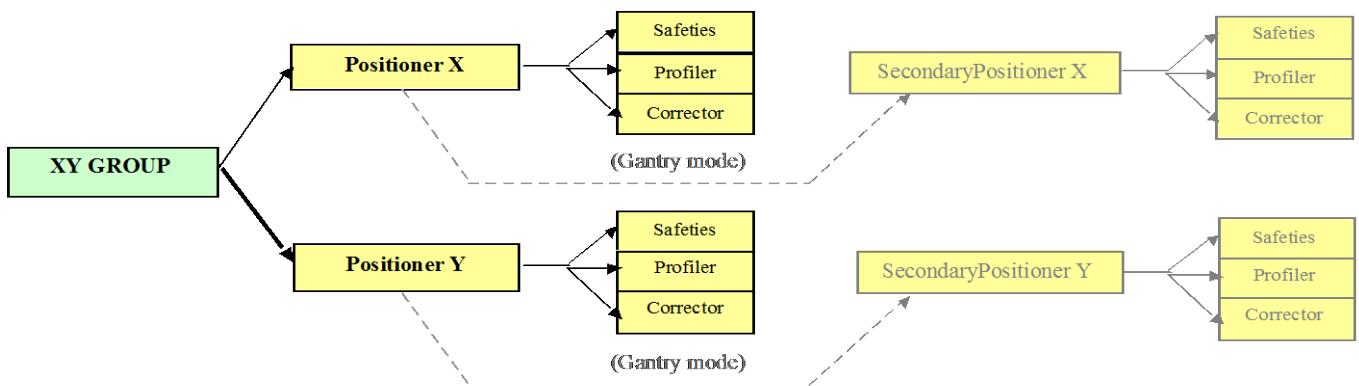
### 3.5 XY Group

An XY group is composed of two positioners, typically in an orthogonal XY configuration.

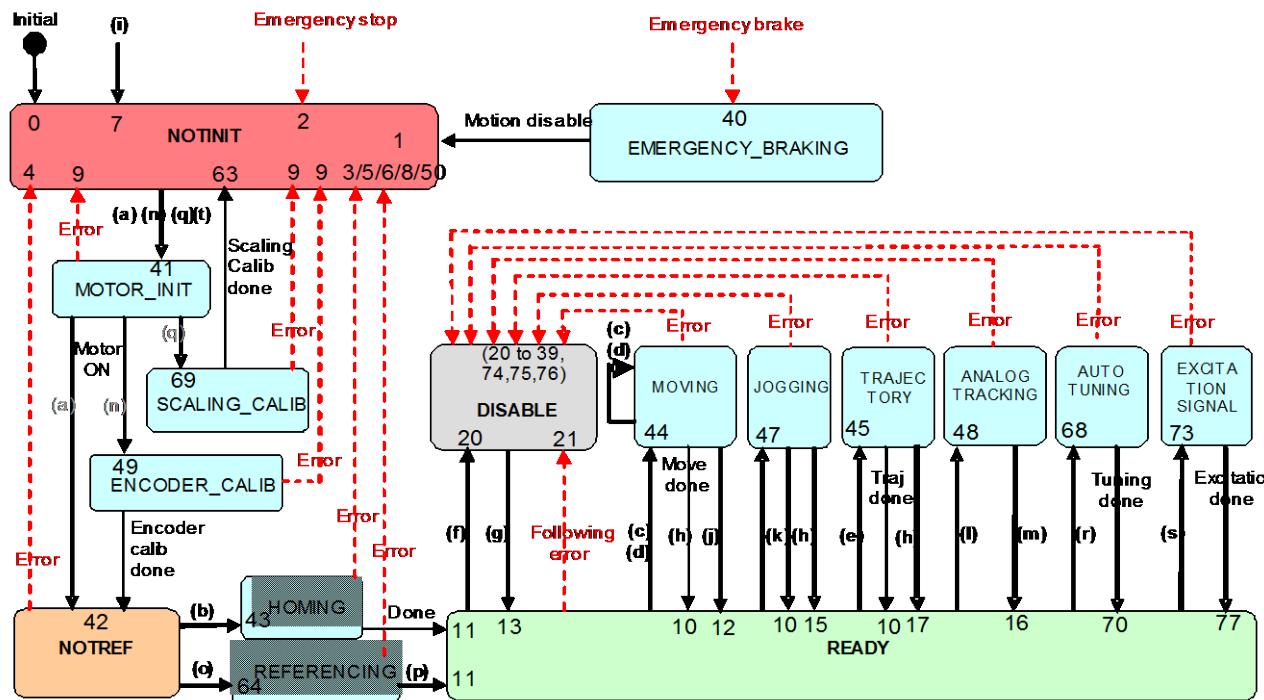
An XY group can be used in GANTRY mode (dual positioner for X or for Y).

It includes an XY mapping feature:  $XY = f(XY)$

It includes a line-arc and a PVT (PositionVelocityTime) two-dimension trajectories.



### 3.5.1 State Diagram



#### Called functions:

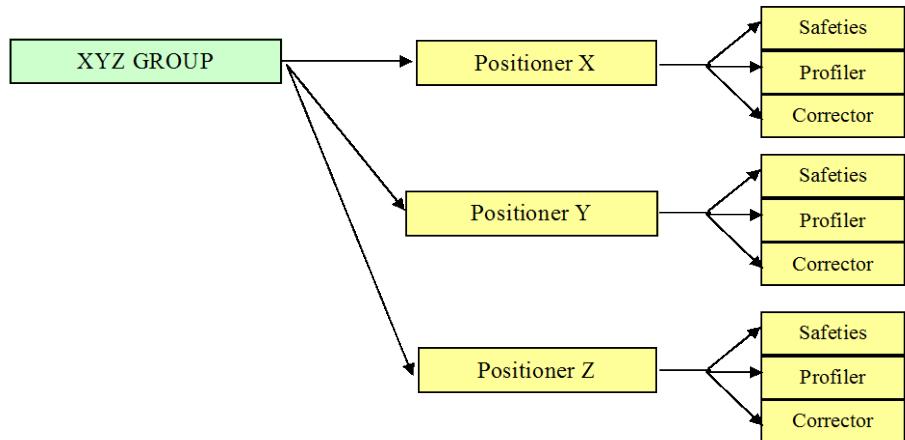
- |     |                      |     |   |
|-----|----------------------|-----|---|
| (a) | GroupInitialize      | (k) | GroupJogModeDisable   |
| (b) | GroupHomeSearch      | (l) | GroupAnalogTrackingModeEnable   |
| (c) | GroupMoveAbsolute    | (m) | GroupAnalogTrackingModeDisable  |
| (d) | GroupMoveRelative    | (n) | GroupInitializeWithEncoderCalibration                                     |
| (e) | XYLineArcExecution   | (o) | GroupReferencingStart   |
| (f) | GroupMotionDisable   | (p) | GroupReferencingStop  |
| (g) | GroupMotionEnable    | (q) | PositionerAccelerationAutoScaling   |
| (h) | GroupMoveAbort       | (r) | PositionerCorrectorAutoTuning   |
| (i) | GroupKill or KillAll | (s) | PositionerExcitationSignalSet / PositionerPreCorrectorExcitationSignalSet |
| (j) | GroupJogModeEnable   | (t) | GroupInitializeNoEncoderReset   |

### 3.6 XYZ Group

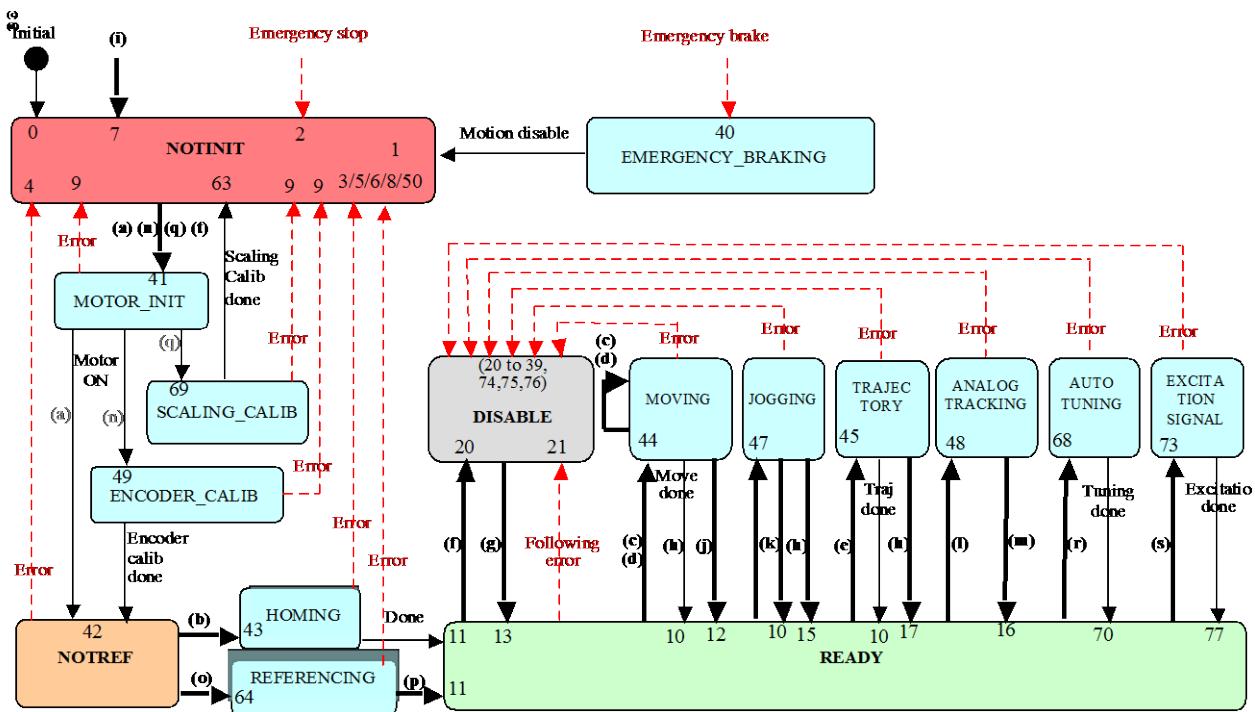
An XYZ group is a three positioner object, typically in an orthogonal XYZ configuration.

It includes an XYZ mapping feature:  $XYZ = f(XYZ)$

It also includes 3D spline trajectories.



### 3.6.1 State Diagram



#### Called functions:

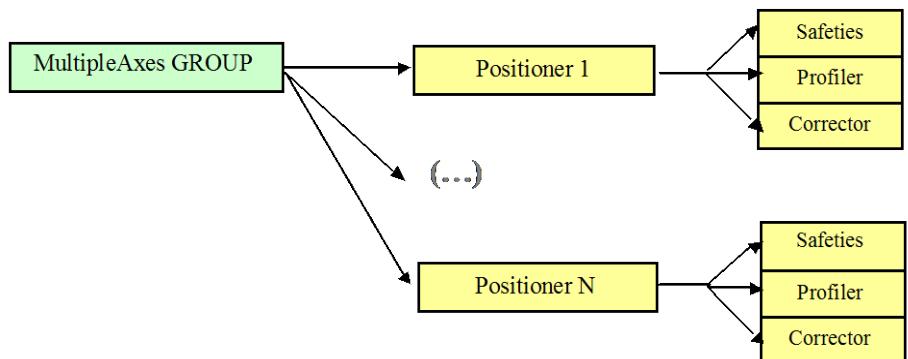
- |     |                      |     |                                       |
|-----|----------------------|-----|---------------------------------------|
| (a) | GroupInitialize      | (k) | GroupJogModeDisable                   |
| (b) | GroupHomeSearch      | (l) | GroupAnalogTrackingModeEnable         |
| (c) | GroupMoveAbsolute    | (m) | GroupAnalogTrackingModeDisable        |
| (d) | GroupMoveRelative    | (n) | GroupInitializeWithEncoderCalibration |
| (e) | XYZSplineExecution   | (o) | GroupReferencingStart                 |
| (f) | GroupMotionDisable   | (p) | GroupReferencingStop                  |
| (g) | GroupMotionEnable    | (q) | PositionerAccelerationAutoScaling     |
| (h) | GroupMoveAbort       | (r) | PositionerCorrectorAutoTuning         |
| (i) | GroupKill or KillAll | (s) | PositionerExcitationSignalSet         |
| (j) | GroupJogModeEnable   | (t) | GroupInitializeNoEncoderReset         |

### 3.7 MultipleAxes Group

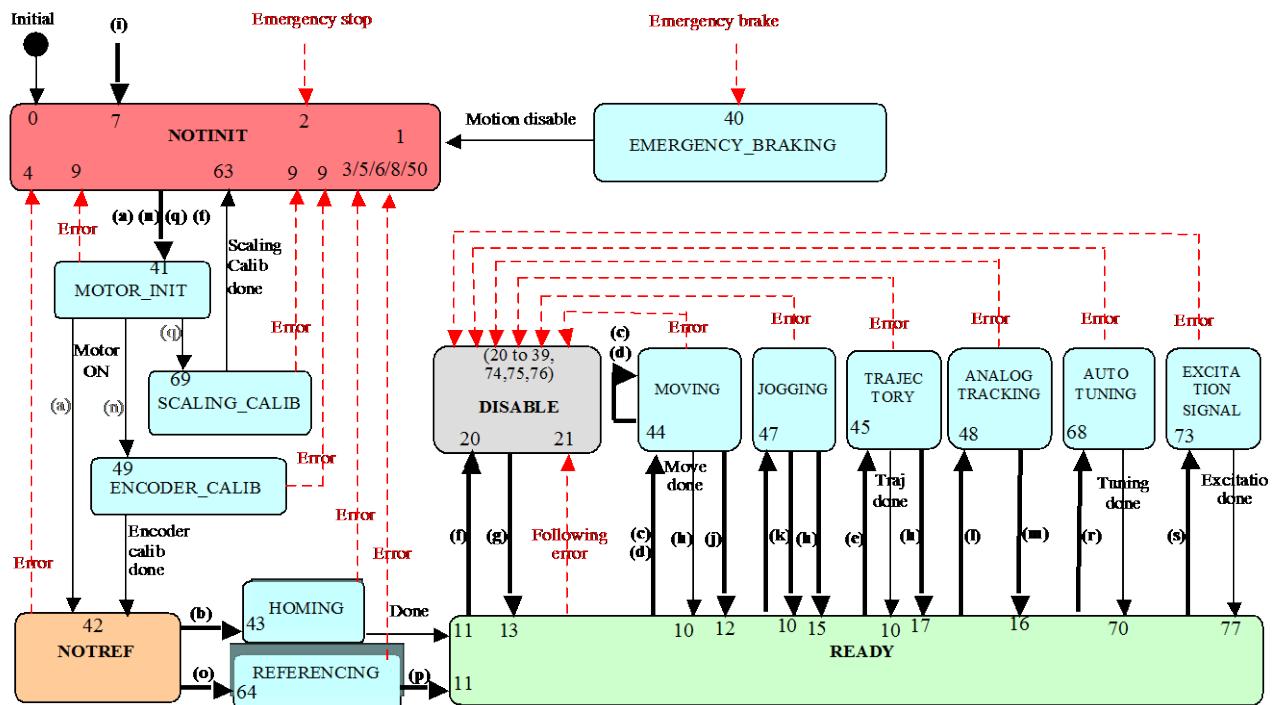
A MultipleAxes group is an n-positioner object, where n can be any number from 1 to 8.

A MultipleAxes group can be used in GANTRY mode (dual positioner for one or several positioners).

It includes the PVT (PositionVelocityTime) and PT (PositionTime) trajectories.



### 3.7.1 State Diagram



#### Called functions:

- |     |                          |     |                                       |
|-----|--------------------------|-----|---------------------------------------|
| (a) | GroupInitialize          | (k) | GroupJogModeDisable                   |
| (b) | GroupHomeSearch          | (l) | GroupAnalogTrackingModeEnable         |
| (c) | GroupMoveAbsolute        | (m) | GroupAnalogTrackingModeDisable        |
| (d) | GroupMoveRelative        | (n) | GroupInitializeWithEncoderCalibration |
| (e) | MultipleAxesPVTExecution | (o) | GroupReferencingStart                 |
| (f) | GroupMotionDisable       | (p) | GroupReferencingStop                  |
| (g) | GroupMotionEnable        | (q) | PositionerAccelerationAutoScaling     |
| (h) | GroupMoveAbort           | (r) | PositionerCorrectorAutoTuning         |
| (i) | GroupKill or KillAll     | (s) | PositionerExcitationSignalSet         |
| (j) | GroupJogModeEnable       | (t) | GroupInitializeNoEncoderReset         |

## 3.8 Analog and Digital I/O

### 3.8.1 GPIO Name List

#### 3.8.1.1 ISA Hardware

##### Digital inputs

<b>GPIO1.DI</b>	Digital Input of the I/O board connector #1 (8 bits)
<b>GPIO2.DI</b>	Digital Input of the I/O board connector #2 (6 bits)
<b>GPIO3.DI</b>	Digital Input of the I/O board connector #3 (6 bits)
<b>GPIO4.DI</b>	Digital Input of the I/O board connector #4 (16 bits)

##### Digital outputs

<b>GPIO1.DO</b>	Digital Output of the I/O board connector #1 (8 bits)
<b>GPIO3.DO</b>	Digital Output of the I/O board connector #3 (6 bits)
<b>GPIO4.DO</b>	Digital Output of the I/O board connector #4 (16 bits)

##### Analog inputs

<b>GPIO2.ADC1</b>	Analog Input #1 of the I/O board connector #2
<b>GPIO2.ADC2</b>	Analog Input #2 of the I/O board connector #2
<b>GPIO2.ADC3</b>	Analog Input #3 of the I/O board connector #2
<b>GPIO2.ADC4</b>	Analog Input #4 of the I/O board connector #2

##### Analog outputs

<b>GPIO2.DAC1</b>	Analog Output #1 of the I/O board connector #2
<b>GPIO2.DAC2</b>	Analog Output #2 of the I/O board connector #2
<b>GPIO2.DAC3</b>	Analog Output #3 of the I/O board connector #2
<b>GPIO2.DAC4</b>	Analog Output #4 of the I/O board connector #2

### 3.8.1.2 PCI Hardware

#### Digital inputs

	<b>Basic GPIO board</b>	<b>Extended GPIO board</b>
<i>GPIO Board #1</i>	GPIO1.DI (8 bits)	GPIO3.DI (8 bits) GPIO5.DI (16 bits) GPIO6.DI (16 bits)
<i>GPIO Board #2</i>	GPIO12.DI (8 bits)	GPIO32.DI (8 bits) GPIO52.DI (16 bits) GPIO62.DI (16 bits)
<i>GPIO Board #3</i>	GPIO13.DI (8 bits)	GPIO33.DI (8 bits) GPIO53.DI (16 bits) GPIO63.DI (16 bits)
<i>GPIO Board #4</i>	GPIO14.DI (8 bits)	GPIO34.DI (8 bits) GPIO54.DI (16 bits) GPIO64.DI (16 bits)

#### Digital outputs

	<b>Basic GPIO board</b>	<b>Extended GPIO board</b>
<i>GPIO Board #1</i>	GPIO1.DO (8 bits)	GPIO3.DO (8 bits) GPIO5.DO (16 bits) GPIO6.DO (16 bits)
<i>GPIO Board #2</i>	GPIO12.DO (8 bits)	GPIO32.DO (8 bits) GPIO52.DO (16 bits) GPIO62.DO (16 bits)
<i>GPIO Board #3</i>	GPIO13.DO (8 bits)	GPIO33.DO (8 bits) GPIO53.DO (16 bits) GPIO63.DO (16 bits)
<i>GPIO Board #4</i>	GPIO14.DO (8 bits)	GPIO34.DO (8 bits) GPIO54.DO (16 bits) GPIO64.DO (16 bits)

#### Analog inputs

	<b>Basic GPIO board</b>	<b>Extended GPIO board</b>
<i>GPIO Board #1</i>	GPIO2.ADC	GPIO4.ADC
<i>GPIO Board #2</i>	GPIO22.ADC	GPIO42.ADC
<i>GPIO Board #3</i>	GPIO23.ADC	GPIO43.ADC
<i>GPIO Board #4</i>	GPIO24.ADC	GPIO44.ADC

#### Analog outputs

	<b>Basic GPIO board</b>	<b>Extended GPIO board</b>
<i>GPIO Board #1</i>	GPIO2.DAC	GPIO4.DAC
<i>GPIO Board #2</i>	GPIO22.DAC	GPIO42.DAC
<i>GPIO Board #3</i>	GPIO23.DAC	GPIO43.DAC
<i>GPIO Board #4</i>	GPIO24.DAC	GPIO44.DAC

**4.****XPS Extended Firmware Architecture (Contact Newport)****NOTE**

Contact Newport to obtain the extended firmware version that includes the following features.

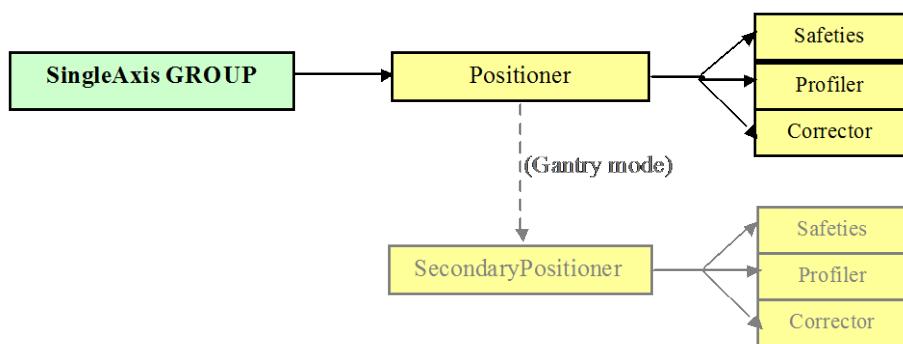
**4.1****SingleAxisWithClamping Group**

The SingleAxisWithClamping Group is composed of one single positioner that allows execution of motion commands.

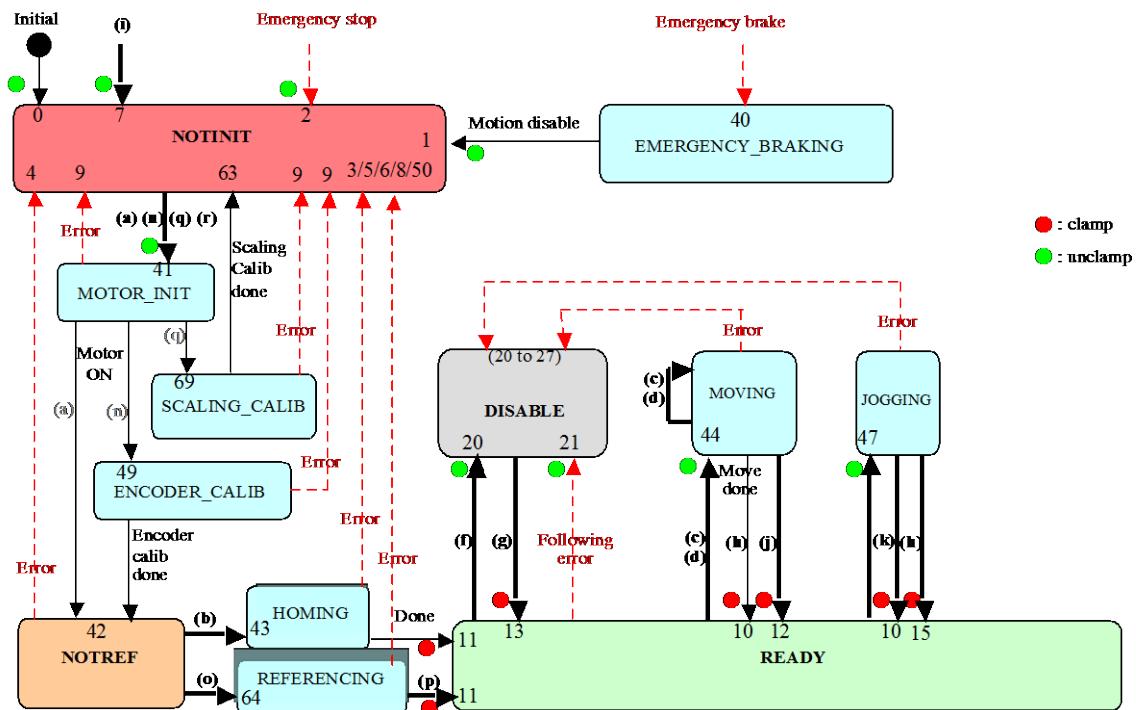
In general, it is similar to SingleAxis group, the only difference is the support of clamping: axis is clamped before moves, unclamped during moves and reclamped at stop. This enhances the stability at a position.

A SingleAxisWithClamping group CANNOT be used in GANTRY mode (*secondary positioner is impossible*).

The XPS controller can handle several SingleAxisWithClamping objects.



### 4.1.1 State Diagram

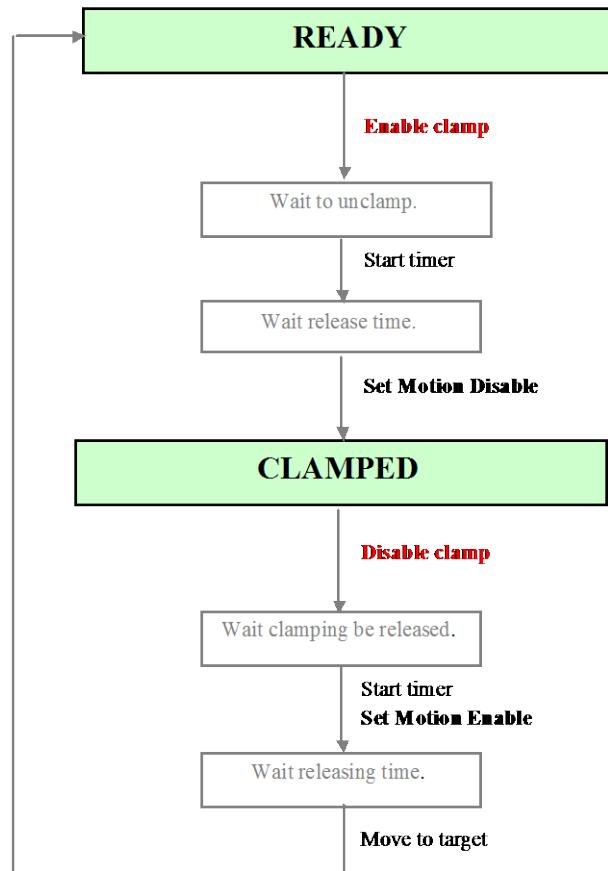


#### Called functions:

- |     |                      |     |                                       |
|-----|----------------------|-----|---------------------------------------|
| (a) | GroupInitialize      | (k) | GroupJogModeDisable                   |
| (b) | GroupHomeSearch      | (l) | GroupAnalogTrackingModeEnable         |
| (c) | GroupMoveAbsolute    | (m) | GroupAnalogTrackingModeDisable        |
| (d) | GroupMoveRelative    | (n) | GroupInitializeWithEncoderCalibration |
| (f) | GroupMotionDisable   | (o) | GroupReferencingStart                 |
| (g) | GroupMotionEnable    | (p) | GroupReferencingStop                  |
| (h) | GroupMoveAbort       | (q) | PositionerAccelerationAutoScaling     |
| (i) | GroupKill or KillAll | (r) | GroupInitializeNoEncoderReset         |
| (j) | GroupJogModeEnable   |     |                                       |

#### 4.1.2 Group Clamping Sequence

##### 4.1.2.1 Clamping state diagram



#### NOTES

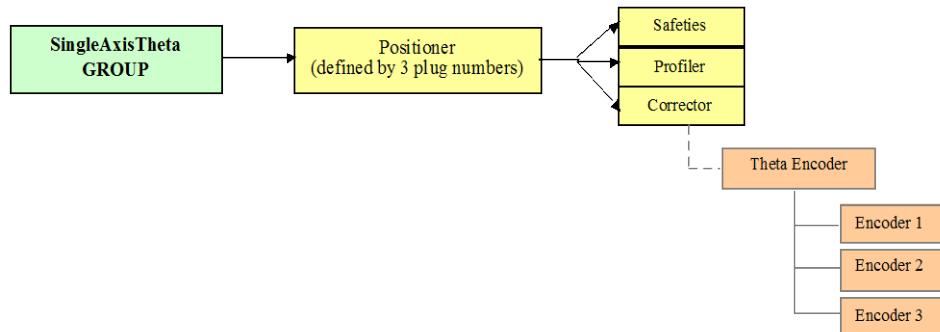
In case of recoverable errors (go to DISABLE state or go to READY state), the clamp is unclamped.

In case of fatal errors (go to NOT INIT state), the clamp stays in the current state (clamped).

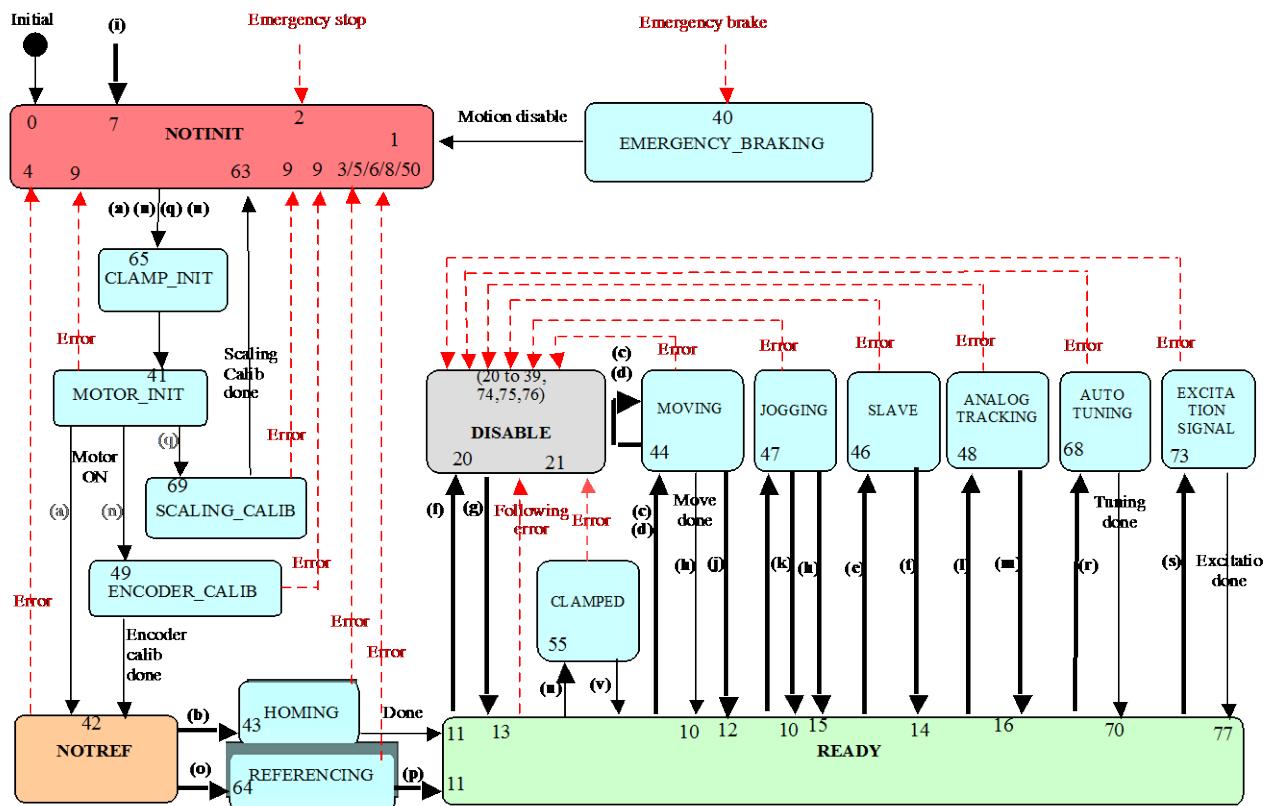
## 4.2 SingleAxisTheta Group

The SingleAxisTheta is composed of one single positioner object with three encoders (Theta encoder) for the execution of motion commands. It includes a “Yaw” mapping and a “Theta” correction on an XY group.

A SingleAxisTheta group CANNOT be used in GANTRY mode (*secondary positioner is impossible*).



### 4.2.1 State Diagram

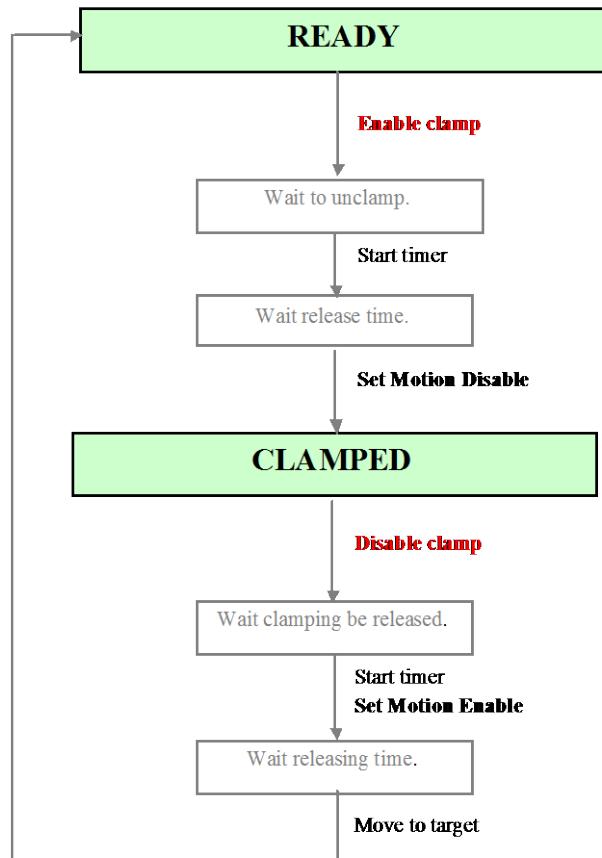


#### Called functions:

- |     |                      |     |   |
|-----|----------------------|-----|---|
| (a) | GroupInitialize      | (l) | GroupAnalogTrackingModeEnable   |
| (b) | GroupHomeSearch      | (m) | GroupAnalogTrackingModeDisable  |
| (c) | GroupMoveAbsolute    | (n) | GroupInitializeWithEncoderCalibration                                     |
| (d) | GroupMoveRelative    | (o) | GroupReferencingStart   |
| (e) | GroupSlaveModeEnable | (p) | GroupReferencingStop  |
| (f) | GroupMotionDisable   | (q) | PositionerAccelerationAutoScaling   |
| (g) | GroupMotionEnable    | (r) | PositionerCorrectorAutoTuning   |
| (h) | GroupMoveAbort       | (s) | PositionerExcitationSignalSet / PositionerPreCorrectorExcitationSignalSet |
| (i) | GroupKill or KillAll | (t) | GroupSlaveModeDisable   |
| (j) | GroupJogModeEnable   | (u) | GroupInitializeNoEncoderReset   |
| (k) | GroupJogModeDisable  |     |   |

## 4.2.2 Group Clamping Sequence

### 4.2.2.1 Clamping state diagram




---

#### NOTES

**In case of recoverable errors (go to DISABLE state or go to READY state), the clamp is unclamped.**

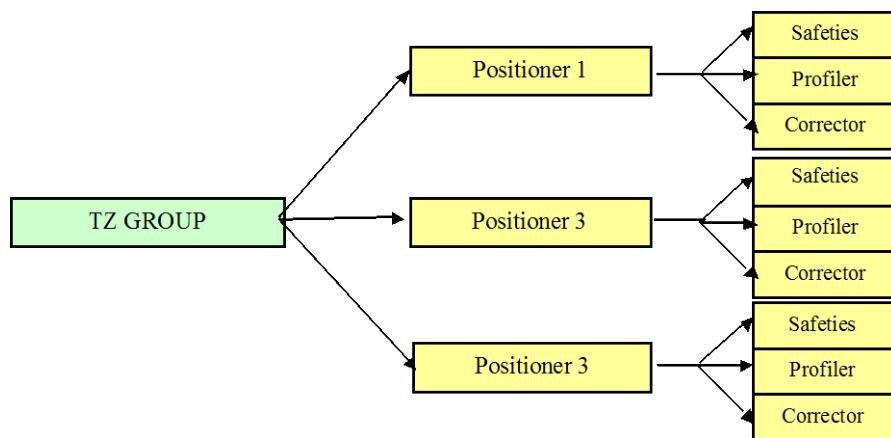
**In case of fatal errors (go to NOT INIT state), the clamp stays in the current state (clamped).**

---

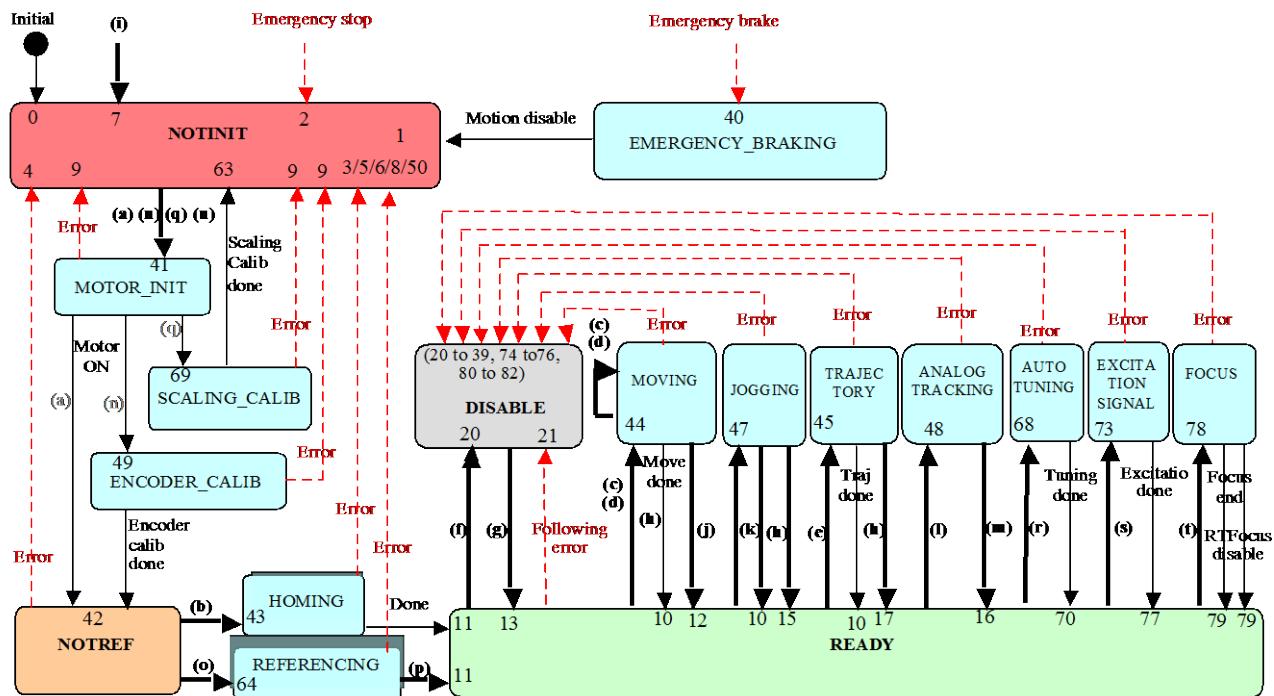
### 4.3 TZ Group

A TZ group is a 3-positioner object, like the XYZ group, but with the following differences:

- It supports the **PVT** trajectories.
- It does not support 3D **spline** trajectories.
- It supports **TZDecoupling**, **XYtoZZZAccelerationFeedforward** and **TZTracking** functionalities.
- It supports **Focus process** via the XPS focus interface board and focus process module (focus.out).
- It has a specific way of initialization  
*(MotorDriverInterface = AnalogAccelerationTZ).*



### 4.3.1 State Diagram



#### Called functions:

- |                              |   |
|------------------------------|---|
| (a) GroupInitialize          | (l) GroupAnalogTrackingModeEnable         |
| (b) GroupHomeSearch          | (m) GroupAnalogTrackingModeDisable        |
| (c) GroupMoveAbsolute        | (n) GroupInitializeWithEncoderCalibration |
| (d) GroupMoveRelative        | (o) GroupReferencingStart                 |
| (e) MultipleAxesPVTExecution | (p) GroupReferencingStop                  |
| (f) GroupMotionDisable       | (q) PositionerAccelerationAutoScaling     |
| (g) GroupMotionEnable        | (r) PositionerCorrectorAutoTuning         |
| (h) GroupMoveAbort           | (s) PositionerExcitationSignalSet         |
| (i) GroupKill or KillAll     | (t) TZFocusModeEnable                     |
| (j) GroupJogModeEnable       | (u) GroupInitializeNoEncoderReset         |
| (k) GroupJogModeDisable      |   |

#### 4.4 User External Module Programming

The user external module programming manages the written by user program blocks (*ExternalModules*) in the XPS controller, with the following conditions:

- Every user external module is written in C language (GNU with QNX Momentics IDE).
- The user ExternalModule (\*.so) must be stored in the “/Admin/UserOptionalModules /” in the XPS controller.
- Every user external module is loaded and executed at boot in adding the external module name in the **SharedLibraryModuleNames** line of the *[GENERAL]* section of system.ref file.

*system.ref*:

**[GENERAL]**

[...]

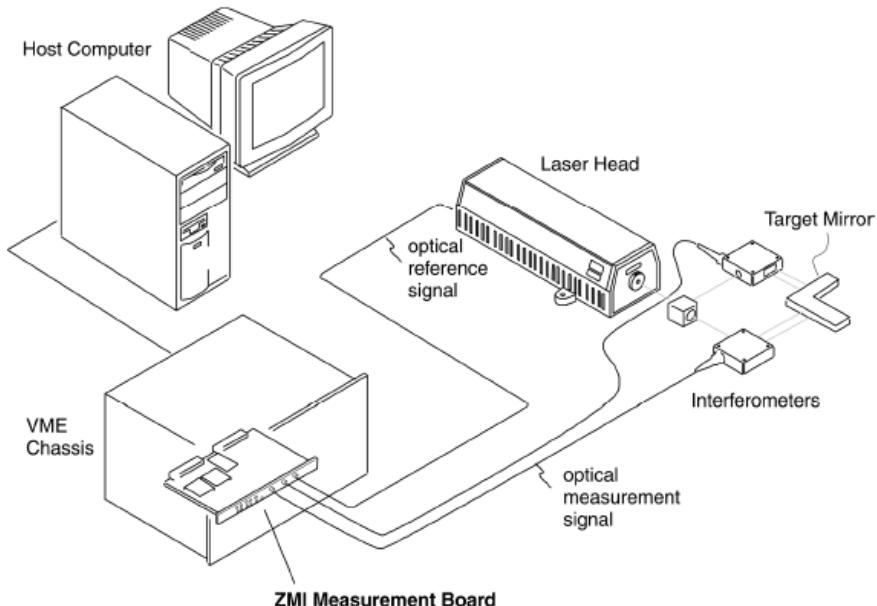
**SharedLibraryModuleNames** = ExternalModule1, ExternalModule2

[...]

## 4.5 ZYGO Interferometer

### 4.5.1 ZMI Measurement System

Refer to **ZMI 2400 series manual - OMP 0537\_F**



### 4.5.2 ZYGO P2 Interface Registers

	P2 Registers	Read	Write
<b>Axis #1</b>	0	Axis status	<i>Read only</i>
	2	Position	
	4	Velocity	
	6	Time	
<b>Axis #2</b>	8	Axis status	<i>Read only</i>
	10	Position	
	12	Velocity	
	14	Time	
<b>General</b>	16	Board status	Command
<b>Info</b>	18	Revision	<i>Read only</i>
	20	ID	

### 4.5.3 Status, Error, and High Nibble P2 Interface Register

This register contains the status bits, error bits, and the 4 high order position data bits. The table below lists the bit positions of these signals in the register. See “Error Detection and Status Reporting” section for more information. The bits that correspond to fatal errors are indicated by “(F)”.

**Table 3-4 Status, Error, and High Nibble P2 Interface Register Bit Positions**

Bit	Description	XPS group state after detection
0	1 (not used)	
1	1 (not used)	
2	1 (not used)	
3	1 (not used)	
4	1 (not used)	
5	1 (not used)	
6	1 (not used)	
7	1 (not used)	
8	1 (not used)	
9	1 (not used)	
10	1 (not used)	
11	User Bit	
12	Position Data bit 32	
13	Position Data bit 33	
14	Position Data bit 34	
15	Position Data bit 35	
16	Reference Signal Present	
17	Reference PLL Locked	
18	System OK	
19	Measurement Signal Present	
20	Position Reset Complete	
21	(F) Reference Signal Missing	Not init
22	(F) Reference PLL Error	Not init
23	(F) System Error	Not init
24	(F) Measure Signal Missing	Not init
25	(F) Measure Signal Dropout	Not init
26	(F) Measure Signal Glitch	Not init
27	(F) Velocity Error	
28	User Velocity Error	
29	(F) Acceleration Error	
30	(F) 36 bit Position Overflow	Not init
31	32 bit Position Overflow	Not init

The high nibble bits are latched at the same time as the P2 Position register, and should be read after the position is read. The error bits (21-31) have the same meaning as the corresponding errors in the VME Error Status register (bits 0-10). The P2 Error register is set and reset separately from the VME Error Status register.

The status bits (16-20) have the same meaning as the corresponding status bits in the VME Status Register (bits 0-4). The *User Bit* shows the state of the *User P2* bit in VME Control Register 1.

#### 4.5.4 ZYGO Error Code Table

Error Code	Description
-100001	Syntax error in command or unrecognized command
-100002	Internal error occurred while parsing request
-100003	Invalid character detected in command input
-100004	Un-terminated string
-100005	Invalid hex number
-100020	Error accessing the Measuring Board
-100021	Operation not supported for ZEC-Measuring Board set
-100022	Invalid Register offset detected
-100023	Invalid axis number detected
-100024	Invalid ADC Mux value detected
-100025	ADC operation timed out
-100026	Invalid EEPROM offset detected
-100027	EEPROM operation timed out
-100028	Invalid register type specified
-100029	Value to write too large for register type
-100030	Value to write missing
-100031	Invalid operation type specified
-100032	Value to write invalid
-100033	Internal error saving Ethernet Card configuration data
-100034	Invalid IP address
-100035	Invalid Subnet mask
-100036	Invalid Gateway IP address
-100037	UDP Measurement Data transmission already in progress
-100050	Another Firmware update is already in progress
-100051	Firmware update irrecoverable file transfer error
-100052	Firmware update file data received is incomplete
-100053	Firmware update error performing a PROM operation

The error description is returned by the “ErrorStringGet” function.

All error code list is returned by the “ErrorListGet” function.

#### 4.5.5 PEG Control Register

The bit assignments of the PEG Control register are listed below. The *PEG Enable Control* and *PEG Disable Control* fields control the overall operation of the PEG subsystem.

Bit	Description
1:0	PEG Pulse Width (00 = 25 ns, 01 = 50 ns, 10 = 75 ns, 11 = 100 ns)
2	Enable P1->P2 operation
3	Enable P2->P1 operation
4	PEG Axis Select (0 = axis 1, 1 = axis 2)
5	PEG Output Enable
6	PEG Output Polarity (0 = Low pulse, 1 = High pulse)
7	Enable Delta 2
10:8	PEG Disable Control 0 = NO_EXIT = Don't disable 1 = P1_EXIT = Disable at P1 exit 2 = P2_EXIT = Disable at P2 exit 3 = ANY_EXIT = Disable at P1 or P2 exit 4 = IMM_EXIT = Disable immediately
11	PEG P2 Delta (0 = Delta 1, 1 = Delta 2)
13:12	PEG Enable Control 0 = NO_ENT = Don't enable 1 = P1_ENT = Enable at P1 entry 2 = P2_ENT = Enable at P2 entry 3 = ANY_ENT = Enable at P1 or P2 entry
15:14	Reserved

#### 4.5.6 ZYGO Axis Error Status List

code	Error Status description	Error mask
0	Success	
0x0001	Reference signal is missing	1
0x0002	Reference PLL Error	1
0x0004	System Error	1
0x0008	Measure Signal Missing	1
0x0010	Measure Signal Dropout	0
0x0020	Measure Signal Glitch	1
0x0040	Velocity Error	1
0x0100	Acceleration Error	0
0x0200	36 bit Position Overflow	1
0x0400	32 bit Position Overflow	1
0x0800	P2 External Sample	0
0x1000	Not used	0
0x2000	PEG Error	0
0x4000	Not used	0
0x8000	IRQ Pending	0

The axis error status register is read via Ethernet. A XPS controller error is generated when one or several bits of Error mask are ON. In this case, the group goes to NOTINIT state.

#### 4.5.7 ZYGO Axis Status List

code	Status description
0	Success
0x0001	Reference signal present
0x0002	Reference PLL
0x0004	System OK
0x0008	Measure Signal present
0x0010	Position Reset Complete
0x0020	ADC Ready
0x0040	ADC Mux = 0
0x0100	External Sample Flag (SCLK0)
0x0200	Not used
0x0400	Not used
0x0800	Not used
0x1000	System Type ST0
0x2000	System Type ST1
0x4000	System Type ST2
0x8000	System Type ST3

The axis status register is read via **Ethernet**.

## 5.

# XPS .NET Software Drivers

---

## 5.1 How to install .NET Drivers for XPS Controller

### 5.1.1 Requirements

.Net Framework is a programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies such as custom desktop applications.

The Windows PC computer requires having at least the .NET Framework 4.5.2 installed and you need to install either 32 bit (x86) or 64 bit (x64) .NET assembly depending on the Windows version you are using.

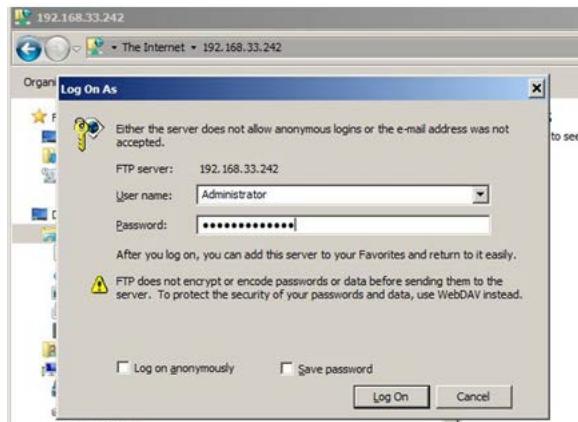
When developing your application, refer to the programming environment documentation to make the installed .NET assembly visible.

To communicate with the XPS controller you will need to:

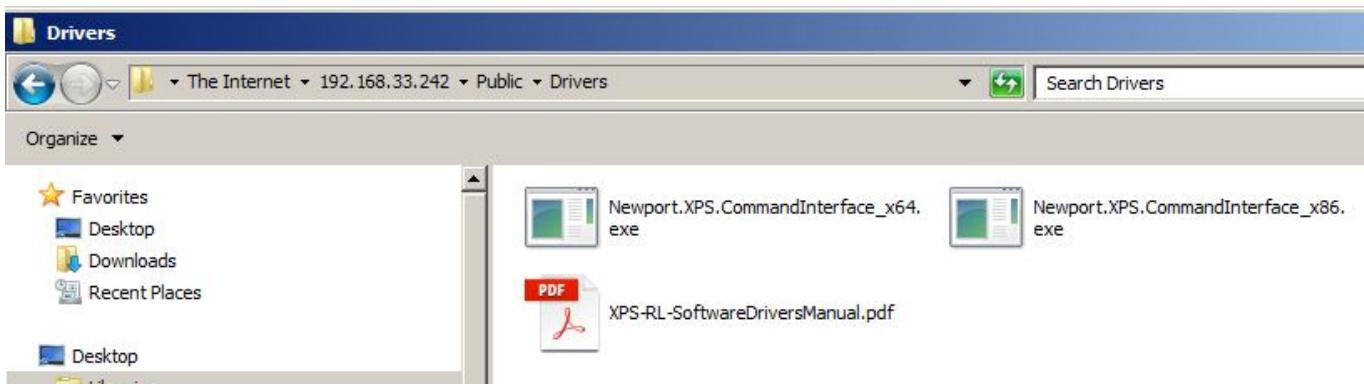
- Use the **OpenInstrument** method to connect to the controller
- Communicate with the controller using any of its API e.g. **FirmwareVersionGet**
- Once your application terminates it needs to disconnect from the controller using the **CloseInstrument** method. If it doesn't close the communication channel and runs many connections to the controller, it can run out of free channels and gets an error.

### 5.1.2 Installing the 32 bit (x86) Windows platform

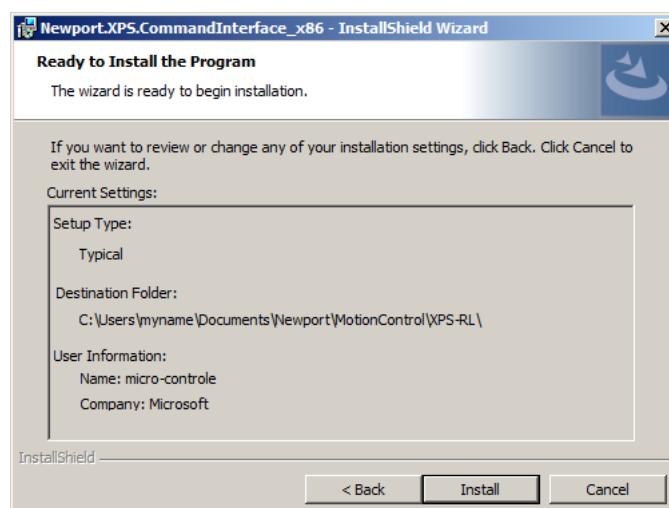
Connect to the XPS controller using the ftp protocol; refer to the XPS user's manual for more details:



Once connected, go to the “/Public/Drivers” folder and download the Newport.XPS.CommandInterface\_x86.exe to your computer:



Once downloaded, run the **Newport.XPS.CommandInterface\_x86** executable file.

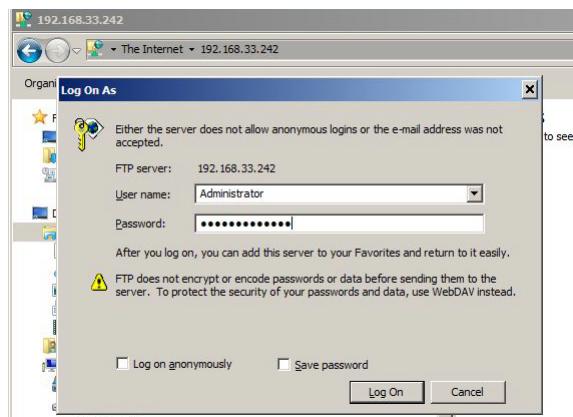


The .Net assembly “Newport.XPS.CommandInterface.dll” V1.0.x.x is installed in the GAC for x86 platforms:

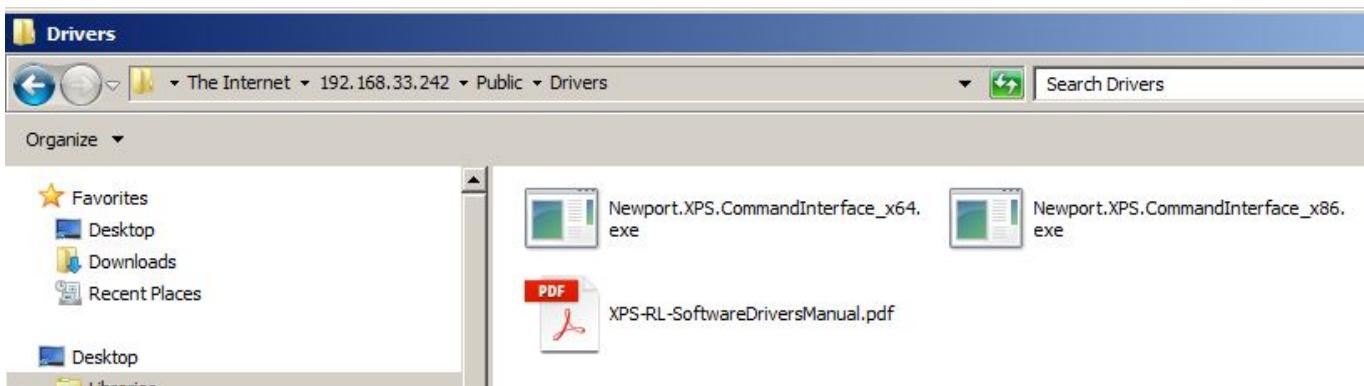
*C:\Windows\Microsoft.NET\assembly\GAC\_32\Newport.XPS.CommandInterface\v4.0\_1.0.0.0\_\_9a267756cf640dcf*

### 5.1.3 Installing the 64 bit (x64) Windows platform

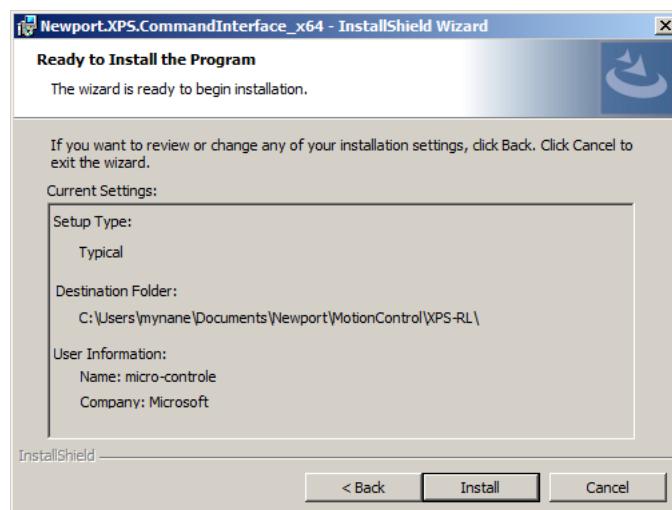
First connect to the XPS through the ftp protocol:



Once connected, go to the “/Public/Drivers” folder and download the Newport.XPS.CommandInterface\_x64.exe to your computer:



Once downloaded, run the **Newport.XPS.CommandInterface\_x64** executable file.



Once installed, the .Net assembly “Newport.XPS.CommandInterface.dll” V1.0.0.0 is located in GAC for x64 platforms:

*C:\Windows\Microsoft.NET\assembly\GAC\_64\Newport.XPS.CommandInterface\v4.0\_1.0.0.0\_\_9a267756cf640dcf*

## 5.2 How to Use XPS .NET Assembly from Visual Studio C#?

Refer to [Microsoft](#) for more information on how to load and use a .NET assembly depending on your Visual Studio version.

### 5.2.1 Add Reference to .NET Assembly

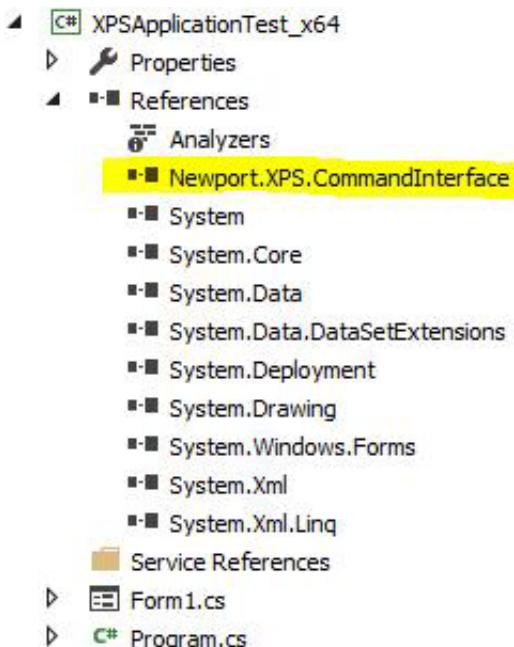
Add Newport.XPS.CommandInterface.dll in References to your project:

#### x86:

*C:\Windows\assembly\GAC\_32\Newport.XPS.CommandInterface\1.0.0.0\_\_9a267756cf640dcf*

#### x64:

*C:\Windows\assembly\GAC\_64\Newport.XPS.CommandInterface\1.0.0.0\_\_9a267756cf640dcf*



## 5.2.2 C# Code Sources

### C# Header

```
using CommandInterfaceXPS; // Newport.XPS.CommandInterface .NET Assembly  
access
```

### Add a Variable to Declare an “XPS” Object

```
CommandInterfaceXPS.XPS m_xpsInterface = null;
```

### Create an Instance of “XPS” Object

```
m_xpsInterface = new CommandInterfaceXPS.XPS();  
if (m_xpsInterface != null)  
...
```

### Open XPS Connection

```
if (m_xpsInterface != null)  
int returnValue = m_xpsInterface.OpenInstrument(m_IPAddress, m_IPPort,  
DEFAULT_TIMEOUT);
```

### Call “XPS” Functions

```
if (m_xpsInterface != null)  
{  
    string XPSVersion = string.Empty;  
    string errorString = string.Empty;  
    int result = m_xpsInterface.FirmwareVersionGet(out XPSVersion, out  
errorString);  
    if (result == CommandInterfaceXPS.XPS.FAILURE)  
    ...  
}
```

### Close XPS Connection

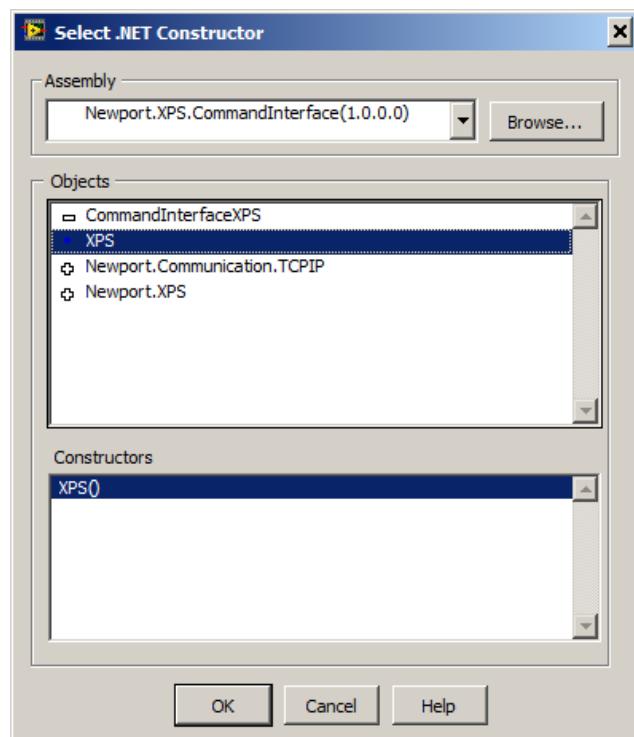
```
if (m_xpsInterface != null)  
    m_xpsInterface.CloseInstrument();
```

## 5.3 How to use XPS .NET Assembly from LabVIEW?

Refer to [LabVIEW](#) for more information on how to load and use a .NET assembly depending on your LabVIEW version.

### 5.3.1 Add Reference to .NET Assembly

Select **CommandInterfaceXPS** and **XPS** constructor from a **.Net Constructor Node** (refer to Connectivity panel):



### 5.3.2 LabVIEW Code Sources

The instance of “XPS” object is created after configuration of .Net Constructor Node:

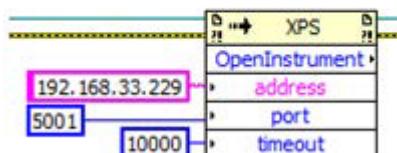
> Connectivity > .NET >

.Net Contructor Node



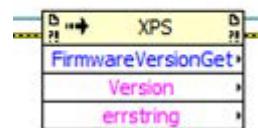
Open XPS connection (Use a .Net Invoke Node to select the XPS method “OpenInstrument”):

.Net Invoke Node



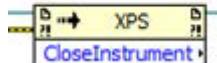
Call “XPS” functions (Use a .Net Invoke Node to select a XPS method):

.Net Invoke Node



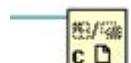
Close XPS connection (Use a .Net Invoke Node to select the XPS method “CloseInstrument”):

.Net Invoke Node



Close .NET Reference:

Close .Net reference



## 5.4 How to use XPS .NET assembly from IronPython?

Refer to [IronPython](#) for more information on how to load and use a .NET assembly depending on your IronPython version.

### 5.4.1 Add Reference to .NET Assembly

Add Newport.XPS.CommandInterface.dll in References of your script:

**x86:**

**import sys**

```
sys.path.append(r'C:\Windows\Microsoft.NET\assembly\GAC_32\Newport.XPS.Comma  
ndInterface\v4.0_1.0.0.0__9a267756cf640dcf')
```

**x64:**

**import sys**

```
sys.path.append(r'C:\Windows\Microsoft.NET\assembly\GAC_64\Newport.XPS.Comma  
ndInterface\v4.0_1.0.0.0__9a267756cf640dcf')
```

### 5.4.2 IronPython Code Source

#### IronPython Header

```
# The CLR module provide functions for interacting with the underlying  
# .NET runtime  
import clr  
  
# Add reference to assembly and import names from namespace (IronPython)  
clr.AddReferenceToFile("Newport.XPS.CommandInterface.dll") from  
CommandInterfaceXPS import *
```

#### Create an Instance

```
# Create XPS interface myXPS = XPS()
```

#### Open XPS Connection

```
def XPS_Open (address, port):  
    # Create XPS interface  
    myXPS = XPS()  
    # Open a socket  
    timeout = 1000  
    result = myXPS.OpenInstrument(address, port, timeout)  
    if result == 0:  
        print 'Open ', address, ":", port, " => Successful"  
    else:  
        print 'Open ', address, ":", port, " => failure ", result  
    return myXPS
```

### **Call XPS Functions**

```
def XPS_GetControllerVersion (myXPS, flag):
    result, version, errString = myXPS.FirmwareVersionGet()
    if flag == 1:
        if result == 0:
            print 'XPS firmware version => ', version
        else:
            print 'FirmwareVersionGet Error => ',errString
    return result, version

def XPS_GetControllerState (myXPS, flag):
    result, state, errString = myXPS.ControllerStatusGet()
    if flag == 1:
        if result == 0:
            print 'XPS controller state => ', state
        else:
            print 'ControllerStatusGet Error => ',errString
    return result, state
```

### **Close XPS Connection**

```
def XPS_Close(myXPS):
    myXPS.CloseInstrument()
```

## 5.5 How to Use XPS .NET Assembly from Matlab?

Refer to [Matlab](#) for more information on how to load and use a .NET assembly depending on your Matlab version.

### 5.5.1 Add Reference to .NET Assembly

```
% Make the assembly visible from Matlab  
asmInfo = NET.addAssembly('Newport.XPS.CommandInterface')
```

### 5.5.2 Matlab Code Source

#### Create an Instance

```
% Make the instantiation myxps=CommandInterfaceXPS.XPS();
```

#### Open XPS Connection

```
% Connect to the XPS controller  
code=myxps.OpenInstrument('192.168.254.254',5001,1000);
```

#### Call XPS Functions

```
% Use API's  
[code Version]=myxps.FirmwareVersionGet [code]=myxps.GroupKill('Group1')  
[code]=myxps.GroupInitialize('Group1')  
[code]=myxps.GroupHomeSearch('Group1')
```

#### Close XPS Connection

```
% Disconnect from the XPS controller  
code=myxps.CloseInstrument;
```

## 6. XPS Functions Description

---

### 6.1 Input Tests Common to all XPS Functions

For all commands, general input tests are the following:

#### General:

- 20 Controller initialization failed.
- 21 XPS initialization in progress.

#### Check the command format:

- 7 Wrong format in the command string.

#### Check the number of parameters:

- 9 Wrong number of parameters in the command.

#### Check input/output parameter type:

- 10 Wrong parameter type in the command string.
- 11 Wrong parameters type in the command string: word or word \* expected.
- 12 Wrong parameter type in the command string: bool or bool \* expected.
- 13 Wrong parameter type in the command string: char \* expected.
- 14 Wrong parameter type in the command string: double or double \* expected.
- 15 Wrong parameter type in the command string: int or int \* expected.
- 16 Wrong parameter (value < 0) or wrong type in the command string: unsigned int or unsigned int \* expected.
- 128 Wrong parameter (value < 0) or wrong type in the command string: unsigned short or unsigned short \* expected.
- 129 Wrong parameter (value < 0) or wrong type in the command string: unsigned long or unsigned long \* expected.
- 132 Wrong parameter (value < 0) or wrong type in the command string: unsigned long long or unsigned long long \* expected.

## 6.2 XPS Functions Lists

### 6.2.1 Standard Functions

#### 6.2.1.1 CleanCoreDumpFolder

##### Name

**CleanCoreDumpFolder** – Clean the folder “/Admin/Public/CoreDump” to delete all debug core dump (\*.core) files.

##### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check if “/Admin/Public/CoreDump” exists: (-22)
- Check error code returned by shell system command: (-100)

##### Description

This function cleans the “/Admin/Public/CoreDump” folder to delete all core files and to free memory.

##### Prototype

```
int CleanCoreDumpFolder(  
    int SocketID  
)
```

##### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

##### Output parameters

None.

##### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -22: Not allowed action.
- -100: Internal error.

### 6.2.1.2 CleanTmpFolder

#### Name

**CleanTmpFolder** – Clean the folder “tmp” to delete all temporary files.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check error code returned by shell system command: (-100)

#### Description

This function cleans the “tmp” folder to delete all temporary files.

#### Prototype

```
int CleanTmpFolder(int SocketID)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -100: Internal error.

### 6.2.1.3 CloseAllOtherSockets

#### Name

**CloseAllOtherSockets** – Closes all sockets beside the one used.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check “Administrator” rights: (-107)

#### Description

This function allows an administrator to close all user sockets except the socket used to call this function.

All other user sockets are closed. So, ERR\_SOCKET\_CLOSED\_BY\_ADMIN error is sent to each running (on an user socket) process before the socket is really closed.

#### Prototype

```
int CloseAllOtherSockets()
```

#### Input parameters

None.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -107: This function requires Administrator rights.

---

#### NOTE

Call the “Login” function to identify the user as “Administrator”.

---

---

#### CAUTION



If some TCL scripts are in progress (after a “TCLScriptExecute” function or a “TCLScriptExecuteAndWait” function), do not use CloseAllOtherSockets to function to kill these TCL scripts. Use the “TCLScriptKill” function to stop the TCL execution and only after can you use the “CloseAllOtherSockets” function to close sockets.

---

### 6.2.1.4 **ControllerMotionKernelTimeLoadGet**

#### Name

**ControllerMotionKernelTimeLoadGet** – Returns controller motion kernel time load.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".

#### Description

This function gets the last exact value of the controller's motion kernel time load (total , corrector, profier and servitudes calculation time).

$$\text{CorrectorTimeLoad} = \text{CorrectorCalculationTime}/\text{CorrectorISRPeriod}$$

$$\begin{aligned}\text{ProfilerTimeLoad} &= \text{ProfilerCalculationTime}/\text{CorrectorISRPeriod} \\ &/\text{ProfileGeneratorISRRatio}\end{aligned}$$

$$\begin{aligned}\text{ServitudesTimeLoad} &= \text{ServitudesCalculationTime}/\text{CorrectorISRPeriod} \\ &/\text{ServitudesISRRatio}\end{aligned}$$

$$\text{TotalTimeLoad} = \text{CorrectorTimeLoad} + \text{ProfilerTimeLoad} + \text{ServitudesTimeLoad}$$

#### **NOTE**

Refer to system.ref file to get **CorrectorISRPeriod**, **ProfileGeneratorISRRatio** and **ServitudesISRRatio**.

#### Prototype

```
int ControllerMotionKernelTimeLoadGet(
    int SocketID,
    double * CPUTotalLoadRatio,
    double * CPUCorrectorLoadRatio,
    double * CPUProfilerLoadRatio,
    double * CPUServitudesLoadRatio
)
```

#### Input parameters

SocketID                    int                    Socket identifier used in each function.

#### Output parameters

CPUTotalLoadRatio	double *	Controller motion kernel total CPU time load.
CPUCorrectorLoadRatio	double *	Controller motion kernel corrector CPU time load.
CPUProfilerLoadRatio	double *	Controller motion kernel profiler CPU time load.
CPUServitudesLoadRatio	double *	Controller motion kernel servitudes CPU time load.

#### Return (In addition to the results of testing the inputs of functions)

- 0:                    No error.

### 6.2.1.5 ControllerRTTimeGet

#### Name

**ControllerRTTimeGet** – Returns the controller corrector period and corrector calculation time.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function gets the last exact value of the controller’s corrector period and the corrector calculation time.

---

#### NOTE

**The default value of XPS controller corrector period is 0.125 ms (corresponding to an 8 kHz controller corrector frequency).**

---

#### Prototype

```
int ControllerRTTimeGet(  
    int SocketID,  
    double * CurrentRTPeriod,  
    double * CurrentRTUsage  
)
```

#### Input parameters

SocketID                    int                    Socket identifier used in each function.

#### Output parameters

CurrentRTPeriod	double *	Controller corrector period (seconds).
CurrentRTUsage	double *	Controller corrector calculation time (seconds).

#### Return (In addition to the results of testing the inputs of functions)

- 0:                    No error.

### 6.2.1.6 ControllerSlaveStatusGet

#### Name

**ControllerSlaveStatusGet** – Returns the slave controller status code.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function, called from the master controller, gets the status of its slave controller(s). The slave controller status codes can take the following values:

- 0: All slave controllers are externally synchronized with the master controller.
- 1: The synchronization cable is not connected to any slave controller.
- 2: At least one slave controller is not externally synchronized with the master controller.

The description of slave controller(s) status codes can be obtained by “ControllerSlaveStatusStringGet” function sent from the master controller.

#### Prototype

```
int ControllerSlaveStatusGet(  
    int SocketID, int * SlaveControllerStatus  
)
```

#### Input parameters

SocketID                    int                    Socket identifier used in each function.

#### Output parameters

SlaveControllerStatus      int \*                    Slave status of the controller.

#### Return (In addition to the results of testing the inputs of functions)

- 0:                    No error.

### 6.2.1.7 ControllerSlaveStatusStringGet

#### Name

**ControllerSlaveStatusStringGet** – Gets the slave controller's status description from a slave controller status code.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".

#### Description

This function returns the slave controller status description corresponding to a slave controller status code.

If the slave status code is not referenced then the function returns (-17) error.

#### Prototype

```
int ControllerSlaveStatusStringGet(  
    int SocketID,  
    int SlaveControllerStatus,  
    char * SlaveControllerStatusString  
)
```

#### Input parameters

SocketID	int	Socket identifier used in each function.
SlaveControllerStatus	int	Slave controller status code.

#### Output parameters

SlaveControllerStatusString char \* Slave controller status description.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.8 ControllerStatusGet

#### Name

**ControllerStatusGet** – Returns the controller status code.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Returns the controller status code. The controller status codes are listed in section 7.10: “Controller Status List”.

The description of the controller status code can be obtained with the “ControllerStatusStringGet” function.

The controller status flag is automatically reset after a controller status reading using the *ControllerStatusGet()* command.

#### Prototype

```
int ControllerStatusGet(  
    int SocketID,  
    int SlaveControllerStatus,  
    char * SlaveControllerStatusString  
)
```

#### Input parameters

SocketID	int	Socket identifier used in each function.
SlaveControllerStatus	int	Slave controller status code.

#### Output parameters

SlaveControllerStatusString char \* Slave controller status description.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

#### **6.2.1.9 ControllerStatusRead**

Name

**ControllerStatusRead** – Reads the controller status code.

## Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

## Description

Returns the controller status code. The controller status codes are listed in section “Group Status List”.

The description of the controller status code can be obtained with the “ControllerStatusStringGet” function.

## Prototype

```
int ControllerStatusRead(int SocketID, int * ControllerStatus)
```

### **Input parameters**

**SocketID** int Socket identifier used in each function.

### Output parameters

**ControllerStatus**      int \*      Status of the controller

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.10 ControllerStatusStringGet

#### Name

**ControllerStatusStringGet** – Gets the controller’s status description from a controller status code.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the controller status description corresponding to a controller status code (see section “Group Status List”).

If the status code is not referenced then the “Unknown controller status code” message will be returned.

#### Prototype

```
int ControllerStatusStringGet(  
    int SocketID,  
    int ControllerStatusCode,  
    char * ControllerStatusString  
)
```

#### Input parameters

SocketID	int	Socket identifier used in each function.
ControllerStatusCode	int	Controller status code.

#### Output parameters

ControllerStatus	char *	Controller status description.
------------------	--------	--------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.11 ControllerSynchronizeCorrectorISR

#### Name

**ControllerSynchronizeCorrectorISR** – Synchronizes the corrector ISR for master-slave controllers system.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function sets the mode of corrector ISR synchronization between master controller and its slave controllers.

Possible synchronization mode (*ModeString*) values:

“*MasterWithEcho*”: Turns a controller in the corrector ISR synchronization mode as Master. The master generates a synchronization signal (derived from its corrector ISR frequency) on a pin of the INHIBIT connector.

“*SlaveOnEcho*”: Turns a controller in the corrector ISR synchronization mode as Slave. The slave receives a synchronization signal (on a pin of INHIBIT connector) coming from its master and uses it as its corrector frequency.

“*Master*”: Return to local mode (each controller generates and uses its own corrector ISR frequency).

---

#### CAUTION



**This function must be called on each controller (master and its slaves) in the following order: *Master first, then 1<sup>st</sup>, 2<sup>nd</sup>... slaves*.**

**Call this function only if every group is in the NOTINIT state.**

**If the controller has just rebooted, wait 300 seconds before calling this function (the necessary time to have the controller corrector ISR frequency stabilized).**

---

#### Prototype

```
int ControllerSynchronizeCorrectorISR(
    int SocketID,
    char * ModeString
)
```

#### Input parameters

SocketID	int	Socket identifier used in each function.
ModeString	char *	Synchronization mode.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.12 DoubleGlobalArrayGet

#### Name

**DoubleGlobalArrayGet** – Get the value of the global array of type “double”.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Verify the index number [0:1000]: (-17)

#### Description

This function gets the variable value from the global array of type “double”, related to a “Number” index. So, the first variable value from the global array is related to the index “0”.

The returned value is returned in a double format.

---

#### NOTE

**The number of data points in the global array of type “double” is limited to 1000.**

---

#### Prototype

```
int DoubleGlobalArrayGet(  
    int SocketID,  
    int Number,  
    double * DoubleValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Number	int	Index in the global array.

#### Output parameters

DoubleValue	double *	Variable value.
-------------	----------	-----------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.13 DoubleGlobalArraySet

#### Name

**DoubleGlobalArraySet** – Set a value for the global array of type “double”.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Verify the index number [0:1000]: (-17)

#### Description

This function sets a new value in the global array located at the “Number” index and the new value is set in a double format.

---

#### NOTE

**The first variable value from the global array is always located at index “0”.**

**The number of data points in the global array is limited to 1000, so the last index is “999”.**

---

#### Prototype

```
int DoubleGlobalArraySet(  
    int SocketID,  
    int Number,  
    double DoubleValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Number	int	Index in the global array.
DoubleValue	double	Variable value.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.14 ElapsedTimeGet

#### Name

**ElapsedTimeGet** – Get the elapsed time since the controller was powered on.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the time in seconds that elapsed since the controller was powered on.

#### Prototype

```
int ElapsedTimeGet(  
    int SocketID,  
    double * ElapsedTime  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

ErrorString	double *	Elapsed time (seconds).
-------------	----------	-------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.15 ErrorStringGet

#### Name

**ErrorStringGet** – Get the error description from a function error code.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

The function returns the error description corresponding to a function error code (see section 7.7: “Positioner Driver Status List”).

If the error code is not referenced then the “Unknown error code” message will be returned.

#### Prototype

```
int ErrorStringGet(int SocketID, int ErrorCode, char * ErrorString)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ErrorCode	int	Error code.

#### Output parameters

ErrorString	char *	Error description.
-------------	--------	--------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.16 EventExtendedAllGet

#### Name

**EventExtendedAllGet** – Returns all “event and action” identifiers in progress.

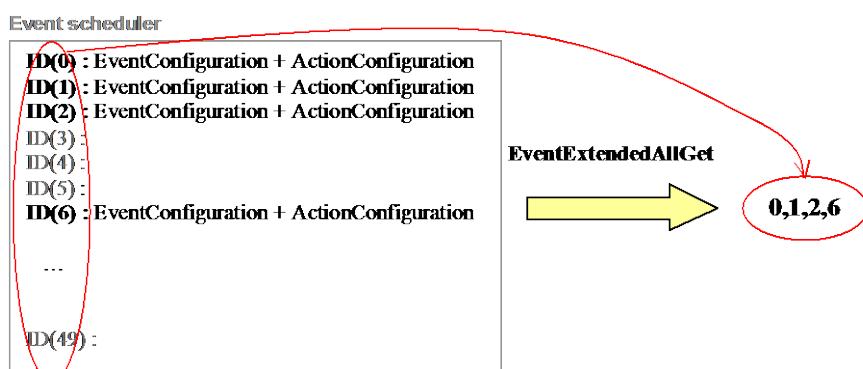
#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Event ID: (-83).

#### Description

Gets the list of all “event and action” combination identifiers from the event scheduler (filled by the **ExtendedEventStart** or **ExtendedEventWait** functions).

The list separator is a comma. If no “event and action” combination is in progress (in the event scheduler) then the error (-83) is returned.



#### Prototype

```
int EventExtendedAllGet(
    int SocketID,
    int EventID,
    char * EventIdentifiersList
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
EventID	int	“Event and action” identifier from “ExtendedEventStart”.

#### Output parameters

EventIdentifiersList	char *	List of “event and action” identifiers in scheduler.
----------------------	--------	--

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -83: Event ID not defined.

### **6.2.1.17 EventExtendedConfigurationActionGet**

#### **Name**

**EventExtendedConfigurationActionGet** – Returns the action combination defined in buffer.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check last action configuration in memory: (-81)

#### **Description**

Returns the combination of action(s) defined by “EventExtendedConfigurationActionSet” function.

If no action is configured in the buffer, (-81) error is returned.




---

#### **NOTE**

This function doesn't return the last activated action. A combination of action(s) can be defined in the buffer but not activated.

---

#### **Prototype**

```
int EventExtendedConfigurationActionGet(
    int SocketID,
    char * ActionConfiguration
)
```

---

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
EventID	int	“Event and action” identifier from “ExtendedEventStart”.

---

#### **Output parameters**

ActionConfiguration	char *	Action combination configured in buffer.
---------------------	--------	--

---

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -81: Action not configured.

### 6.2.1.18 EventExtendedConfigurationActionSet

#### Name

**EventExtendedConfigurationActionSet** – Defines a combination of one or several actions in buffer.

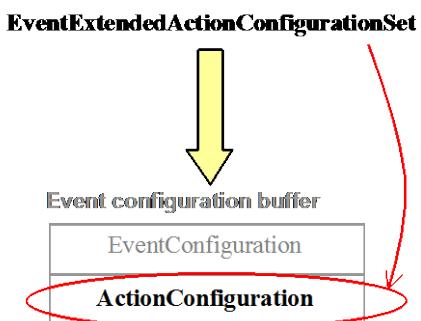
#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Action parameters: (-17), (-8)
- Action to execute: (-32) “Gathering” action.
- Action name: (-39)
- Last action configured in memory: (-81)

#### Description

Defines a combination of one or several actions but does not activate the actions. Use the “EventExtendedStart” function to activate these defined actions. For each action, 4 parameters can be configured ... see event specification to see which are necessary. The actions are defined in section “Events and Actions” in the XPS user’s manual.

The number of actions in a combination is limited to 10 actions.



Refer to section 7.2: “Actions List”.

---

#### NOTE

**Before activating the defined actions, you must configure the events. Only then, can you use the “EventExtendedStart” or “EventExtendedWait” function.**

**For the “ExecuteTCLScript” action, the “ActionParameter3” represents a list of arguments, which must be separated with a semicolon (;).**

---

#### Prototype

```

int EventExtendedConfigurationActionSet(
    int SocketID,
    int NbElements,
    char * ExtendedActionName,
    char * ActionParameter1,
    char * ActionParameter2,
    char * ActionParameter3,
    char * ActionParameter4)
  
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NbElements	int	Number of events in configuration.
ExtendedActionName	char *	Event full name (maximum size = 250). The events are defined in section “Events and Actions” in the XPS user’s manual.
ActionConfiguration	char *	Action combination configured in buffer.
ActionParameter1	char *	optional action’s parameter #1 (maximum size = 250).
ActionParameter2	char *	optional action’s parameter #2 (maximum size = 250).
ActionParameter3	char *	optional action’s parameter #3 (maximum size = 250).
ActionParameter4	char *	optional action’s parameter #4 (maximum size = 250).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -32: Gathering not configured.
- -39: Mnemonic action doesn't exist.

### **6.2.1.19 EventExtendedConfigurationTriggerGet**

#### **Name**

**EventExtendedConfigurationTriggerGet** – Returns the trigger defined in the buffer.

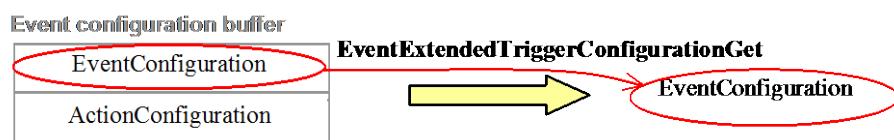
#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Last event configuration in memory: (-80)

#### **Description**

Returns the last event defined in buffer by “EventExtendedConfigurationTriggerSet” function.

If no event is defined in buffer, (-80) error is returned.




---

#### **NOTE**

**This function doesn't return the last activated event. An event can be configured but not activated.**

---

#### **Prototype**

```
int EventExtendedConfigurationTriggerGet(
    int SocketID,
    char * EventTriggerConfiguration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### **Output parameters**

EventTriggerConfiguration    char \*    Event combination configured in buffer.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0:         No error.
- -80:       Event not configured.

### 6.2.1.20 EventExtendedConfigurationTriggerSet

#### Name

**EventExtendedConfigurationTriggerSet** - Defines a combination of one or several events in buffer.

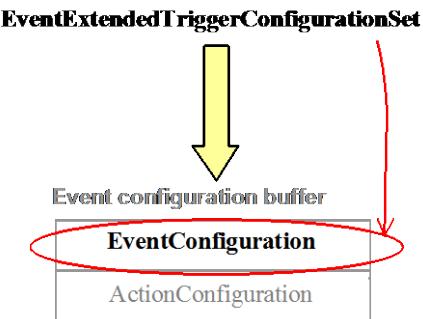
#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Event actor: (-8)
- Event name: (-40)

#### Description

Defines one trigger (combination of one or several events). To activate the trigger, use the "EventExtendedStart" function. For each event, 4 parameters can be configured... see event specification to see the necessary parameters. The events are defined in section "Events and Actions" in the XPS user's manual.

The number of events in a combination is limited to 10 events.



Each full event name is defined as **[actor].[category].event** (see Event list):

[actor] - Optional actor name (Group name, Positioner name, GPIO name or Nothing)

[category] - Optional category name (Event category or Nothing)

event - Event name

---

#### NOTE

**Before activating this event combination, you must define one or several action(s) with the "EventExtendedConfigurationTriggerSet" function. Next, use the "EventExtendedStart" or "EventExtendedWait" function to launch these defined "event and action".**

---

#### Prototype

```

int EventExtendedConfigurationTriggerSet(
    int SocketID,
    int NbElements,
    char * ExtendedEventName,
    char * EventParameter1,
    char * EventParameter2 ,
    char * EventParameter3,
    char * EventParameter4
)
  
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NbElements	int	Number of events in configuration.
ExtendedEventName	char *	list of event full names (maximum size = 250) – separator is ‘;’.
EventParameter1	char *	list of optional event’s parameter #1 (maximum size = 250).
EventParameter2	char *	list of optional event’s parameter #2 (maximum size = 250).
EventParameter3	char *	list of optional event’s parameter #3 (maximum size = 250).
EventParameter4	char *	list of optional event’s parameter #4 (maximum size = 250).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -40: Mnemonic event doesn't exist.

### 6.2.1.21 EventExtendedGet

#### Name

**EventExtendedGet** – Returns the details of “event and action” combinations in scheduler defined by an identifier.

#### Input tests

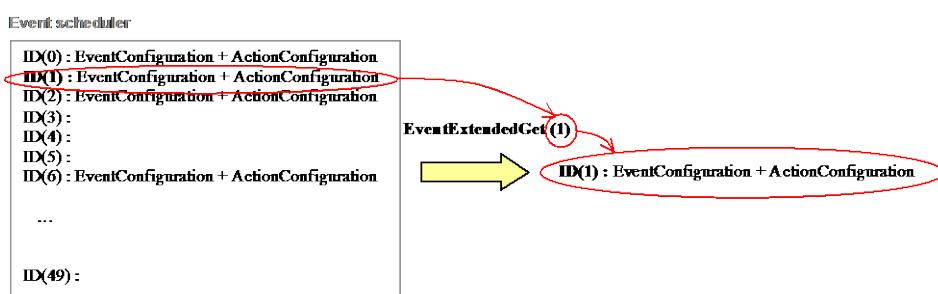
- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Event identifier [0:49]: (-83)

#### Description

Returns the composition of events and actions in progress defined by an identifier. This identifier is defined in the “EventExtendedStart” function.

The identifier must be defined between 0 and 49, if its value is “-1” then it’s not defined.

If the configured event is already deleted, (-83) error is returned.



#### Prototype

```
int EventExtendedGet(
    int SocketID,
    int EventID,
    char * EventConfiguration,
    char * ActionConfiguration
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
EventID	int	“Event and action” identifier from “ExtendedEventStart”.

#### Output parameters

EventConfiguration	char *	Event combination defined in scheduler.
ActionConfiguration	char *	Action combination defined in scheduler.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -83: Event ID not defined.

### 6.2.1.22 EventExtendedRemove

#### Name

**EventExtendedRemove** – Removes an “event and action” combination in the scheduler defined by an identifier.

#### Input tests

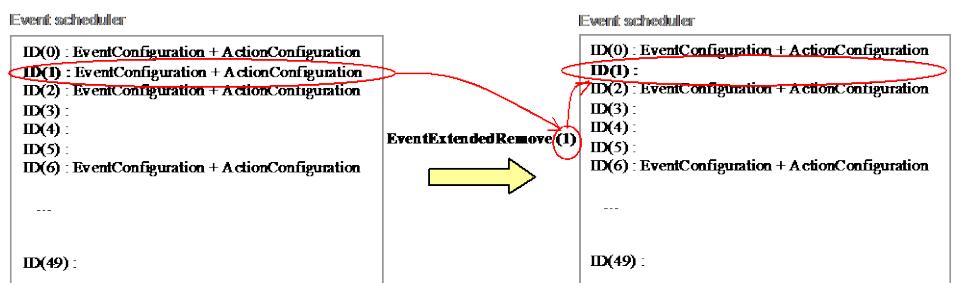
- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Actor event: (-8)
- Event identifier [0:49]: (-17), (-83)

#### Description

Deletes the “event(s) and action(s)” combination in the scheduler defined by an event identifier. This identifier is defined in the “EventExtendedStart” function.

The identifier must be defined between 0 and 49, or -1. If the identifier is equal to “-1”, the EventExtendedRemove function removes all current “event and action” combinations.

If the configured event is already deleted, (-83) is returned.



#### Prototype

```
int EventExtendedRemove(
    int SocketID,
    int EventID
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
EventID	int	“Event and action” identifier.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -83: Event ID not defined.

### 6.2.1.23 EventExtendedStart

#### Name

**EventExtendedStart** – Activates the “event and action” defined in the buffer.

#### Input tests

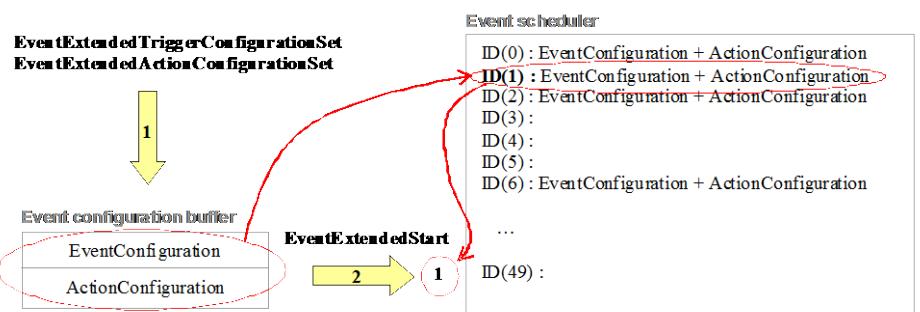
- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Event name to execute: (-8), (-40)
- Last event configuration in memory: (-80)
- Last action configuration in memory: (-81)
- Number of compositions in execution: (-82)

#### Description

Launches the configured event(s) and action(s) from the event configuration buffer into the event scheduler and gets an event identifier. The identifier must be defined between 0 and 49, if its value is “-1” then that means it’s not defined.

If no event is configured in buffer, (-80) error is returned.

If no action is configured in buffer, (-81) error is returned.



#### NOTE

In the event scheduler, when a configured event has occurred it is deleted from the event scheduler.



#### CAUTION

If the configured event is PERMANENT then it is not deleted after it occurs, and must use the “EventExtendedRemove” function to delete it.

#### Prototype

```
int EventExtendedStart(
    int SocketID,
    int EventID
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

### **Output parameters**

EventID                    int \*            “Event and action” identifier.

**Return** (In addition to the results of testing the inputs of functions)

- 0:            No error.
- -8:           Wrong object type for this command.
- -17:          Parameter out of range or incorrect.
- -40:          Mnemonic event doesn't exist.
- -80:          Event not configured.
- -81:          Action not configured.
- -82:          Event buffer is full.

### 6.2.1.24 EventExtendedWait

#### Name

**EventExtendedWait** – Activates the last “event” configuration in memory and wait until it occurs.

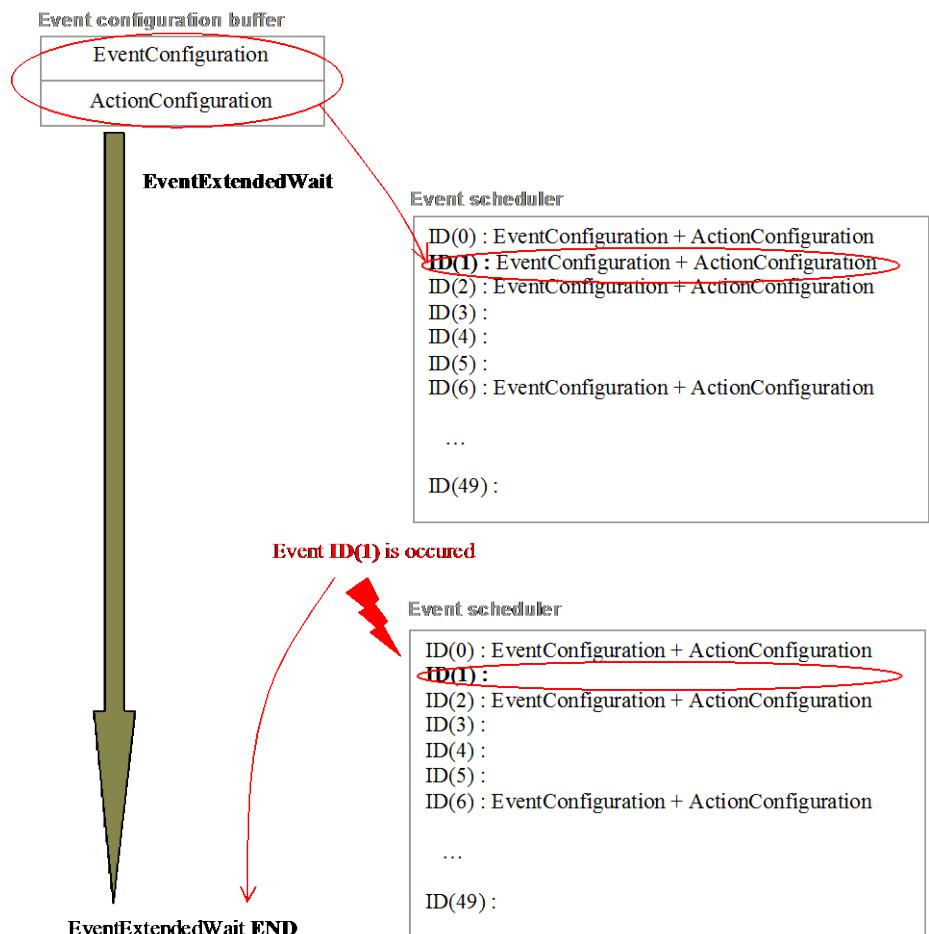
#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Event actor: (-8)
- Last event configuration in memory: (-80)
- Number of compositions in execution: (-82)

#### Description

Launches the last configured event(s) into the event scheduler and wait until it occurs to unlock the socket.

If no “event and action” combination is configured in the event configuration buffer, (-80) error is returned.



#### Prototype

```
int EventExtendedWait(
    int SocketID
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -40: Mnemonic event doesn't exist.
- -80: Event not configured.
- -82: Event buffer is full.

### 6.2.1.25 ExternalModuleFirmwareVersionGet

#### Name

**ExternalModuleFirmwareVersionGet** – Get the firmware version of an ExternalModule.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the module number (must  $\geq 1$  and  $\leq NbExternalModules$ ): (-17)  
Here:  $NbExternalModules$  is the number of modules declared in the line *SharedLibraryModuleNames* of *system.ref*

#### Description

This function gets the firmware version of an ExternalModule.

#### Prototype

```
int ExternalModuleFirmwareVersionGet(
```

```
    int SocketID,  
    int ModuleNumber,  
    char *Version  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ModuleNumber	int	External module number

#### Output parameters

Version	char *	External module firmware version.
---------	--------	-----------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.26 ExternalModuleScanFuncTimeDurationsGet

#### Name

**ExternalModuleScanFuncTimeDurationsGet** – Get the current and the maximum scan executing time for an ExternalModule.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the module number (must  $\geq 1$  and  $\leq NbExternalModules$ ): (-17)
- Here:  $NbExternalModules$  is the number of modules declared in the line *SharedLibraryModuleNames* of *system.ref*

#### Description

An ExternalModule has a scan function that is called periodically by Newport MotionKernel. This function gets the current and the maximum scan executing time for an ExternalModule.

#### Prototype

```
int ExternalModuleScanFuncTimeDurationsGet(  
    int SocketID,  
    int ModuleNumber,  
    double *CurrentDuration,  
    double *MaximumDuration  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ModuleNumber	int	External module number

#### Output parameters

CurrentDuration	char *	Current scan executing duration.
MaximumDuration	char *	Maximum scan executing duration.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.27 ExternalModuleSocketFree

#### Name

**ExternalModuleSocketFree** – Free the socket previously reserved for an ExternalModule.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the module number (must  $\geq 1$  and  $\leq NbExternalModules$ ): (-17)  
Here:  $NbExternalModules$  is the number of modules declared in the line *SharedLibraryModuleNames* of *system.ref*

#### Description

This function frees the socket previously reserved for an ExternalModule. If the function is executed successfully via this socket all the controller functions (*like FirmwareVersionGet()*, *Elapsed Time Get()*, *ErrorStringGet()*, ...) become active, while ExternalModule functions (*like ExternalModuleTZPositionCurrentGet()*, *ExternalModuleGPIODigitalInputGet()*, ...) become inactive.

(*return error -18: Positioner Name doesn't exist or unknown command*).

#### Prototype

```
int ExternalModuleSocketFree(  
    int SocketID,  
    int ModuleNumber  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ModuleNumber	int	External module number

#### Output parameters

None

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.28 **ExternalModuleSocketReserve**

#### Name

**ExternalModuleSocketReserve** – Reserves the current socket for an ExternalModule.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the module number (must  $\geq 1$  and  $\leq NbExternalModules$ ): (-17)  
Here:  $NbExternalModules$  is the number of modules declared in the line *SharedLibraryModuleNames* of *system.ref*
- Check if the current socket is already reserved for another ExternalModule: (-22)
- Check if the ExternalModule has already been linked with another socket: (-22)

#### Description

- To be able to execute the own API functions of an ExternalModule via a TCP Terminal, the user must reserve a socket for this ExternalModule.
- This function reserves the current socket for an ExternalModule. If the function is executed successfully, via this socket the own ExternalModule functions (*like ExternalModuleTZPositionCurrentGet()*, *ExternalModuleGPIODigitalInputGet()*, ...) become functioning, whereas the controller functions (*like FirmwareVersionGet()*, *ElapsedTimeGet()*, *ErrorStringGet()*, ...) become inactive (*return Unknown command*).

#### Prototype

```
int ExternalModuleSocketReserve(
    int SocketID,
    int ModuleNumber
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ModuleNumber	int	External module number

#### Output parameters

None

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.

### 6.2.1.29 FileGatheringRename

#### Name

**FileGatheringRename** – Renames “Gathering.dat” file.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This API renames the “Gathering.dat” file with another .dat file name.

#### Prototype

```
int FileGatheringRename(  
    int SocketID,  
    char * NewFileName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NewFileName	char *	New file name used to rename “Gathering.dat”.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.30 FileScriptHistoryRename

#### Name

**FileScriptHistoryRename** – Renames “history.tcl” file

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This API renames the “history.tcl” file with another tcl file name.

#### Prototype

```
int FileScriptHistoryRename(  
    int SocketID,  
    char * NewFileName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NewFileName	char *	New file name used to rename “history.tcl”.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.31 FirmwareBuildVersionNumberGet

#### Name

**FirmwareBuildVersionNumberGet** – Gets the built firmware version

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function gets the controller name and the firmware version.

Example of returned version string:

“XPS Unified Firmware V1.0.0”

- Controller name is **XPS**.
- Firmware version is **V1.0.0**.

#### Prototype

```
int FirmwareBuildVersionNumberGet(
```

```
    int SocketID,  
    char * Version  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the TCP_ConnectToServer” function.
----------	-----	---

#### Output parameters

Version	char *	Controller firmware version.
---------	--------	------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.32 FirmwareVersionGet

#### Name

**FirmwareVersionGet** – Gets the version of the controller.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function gets the controller version defined in firmware.ref.

Example of returned version string: “XPS-Q8 V2.1.0”

#### Prototype

```
int FirmwareVersionGet(
```

```
    int SocketID,  
    char * Version  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Version	char *	Controller version.
---------	--------	---------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.33 GatheringConfigurationGet

#### Name

**GatheringConfigurationGet** – Returns the current configuration of internally triggered data gathering.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Gathering must be configured: (-32)

#### Description

This function returns the current configuration of internally triggered data gathering.

Use the “GatheringListGet” function to retrieve a complete list of allowed gathering types.

For a more thorough description of the internal data gathering capability, please refer to section Data Gathering/Internal Data Gathering of XPS Motion Tutorial.

#### Prototype

```
int GatheringConfigurationGet(int SocketID, char * TypeList)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

TypeList	char *	List of configured gathering types (separator is semicolon).
----------	--------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -32: Gathering not configured.

### 6.2.1.34 GatheringConfigurationSet

#### Name

**GatheringConfigurationSet** – Configures a data gathering action.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input gathering mnemonic: (-29)
- Gathering must not be in progress: (-43)

#### Description

Defines one or several types of data gathered during the internal triggered data gathering.

Maximum of 1000000 points can be acquired.

Maximum of 25 data types can be configured in a gathering.

Refer to section 7.3: “Gathering Data Types”.

The “GatheringListGet” function can be used to retrieve a complete list of gathering types.

For a more thorough description of the internal data gathering capability, please refer to section Data Gathering/Internal Data Gathering of XPS Motion Tutorial.

#### Prototype

```
int GatheringConfigurationSet(
    int SocketID,
    int NbElements,
    char * TypeArray
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NbElements	int	Number of types.
TypeArray	char *	Array of configured gathering types.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -29: Mnemonic gathering type doesn't exist.
- -32: Gathering not configured.
- -43: Gathering running.

### 6.2.1.35 GatheringCurrentNumberGet

#### Name

**GatheringCurrentNumberGet** – Returns the current and maximum number of gathered data points.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Gathering must be configured: (-32)

#### Description

This function returns the current and maximum number of data points gathered during the internal triggered data gathering.

For more thorough description of the internal data gathering capability, please refer to section Data Gathering/Internal Data Gathering of XPS Motion Tutorial.

#### Prototype

```
int GatheringCurrentNumberGet(  
    int SocketID,  
    int * CurrentNumber,  
    int * MaxSamplesNumber  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	---

#### Output parameters

CurrentNumber	int *	Current number during acquisition.
MaxSamplesNumber	int *	Maximum number of samples.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -32: Gathering not configured.

### 6.2.1.36 GatheringDataAcquire

#### Name

**GatheringDataAcquire** – Manually Acquires one data set.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Gathering must be configured: (-32)
- Gathering must not be in progress: (-43)
- Check gathering buffer size: (-111)

#### Description

This function manually acquires one data set configured by “GatheringConfigurationSet” function.

#### Prototype

```
int GatheringDataAcquire(int SocketID)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -32: Gathering not configured.
- -43: Gathering running.
- -111: Gathering buffer is full.

### 6.2.1.37 GatheringDataGet

#### Name

**GatheringDataGet** – Reads one data line from the current gathering buffer.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check index number: (-17)
  - IndexPoint  $\geq 0$ .
  - IndexPoint <currently gathered data number.
- Check gathering state: (-32)

#### Description

This function reads a line of data from the current gathering buffer. The buffer line number is defined by the index of an acquired point.

The separator is “;” in the returned data line.

Gathering must be configured in order to use this function, otherwise (-32) error is returned.

#### Prototype

```
int GatheringDataGet(int SocketID, int IndexPoint, char * DataBufferLine)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
IndexPoint	int	Index of an acquired data from the current gathering buffer.

#### Output parameters

DataBufferLine	char *	String contains values from the current buffer at the selected index.
----------------	--------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -32: Gathering not configured.

### 6.2.1.38 GatheringDataMultipleLinesGet

#### Name

**GatheringDataMultipleLinesGet** – Reads several data lines from the current gathering buffer in memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check index number: (-17)
  - IndexPoint  $\geq 0$  (**Note:** index #0 = line #1)
  - IndexPoint <currently gathered data number.
- Check gathering state: (-32)

#### Description

This function reads one or several data lines from the current gathering buffer. The buffer line number is defined by the index of an acquired point.

The separator is “;” in the returned data line and the end of each line is carriage return “\n”.

Gathering must be configured in order to use this function, otherwise (-32) error is returned.

*Example of gathering buffer in memory:*

index	Data1	Data2	Data3	Data4	Data5
0 →	1	10	0.1	21	100
1 →	2	20	0.2	22	102
2 →	3	30	0.3	23	103
3 →	4	40	0.4	24	104
5 →	5	50	0.5	25	105

GatheringDataMultipleLinesGet(0, 3, myString)

=>0 = the start line is #1

=>3 = the number of lines to read is 3

=>myString = buffer to get the part of buffer (32767 characters maximum)

index	Data1	Data2	Data3	Data4	Data5
0 →	1	10	0.1	21	100
1 →	2	20	0.2	22	102
2 →	3	30	0.3	23	103
3 →	4	40	0.4	24	104
5 →	5	50	0.5	25	105

“myString” result:

1;10;0.1;21;100

2;20;0.2;22;102

3;30;0.3;23;103

**GatheringDataMultipleLinesGet(1, 4, myString)**

= >1 = the start line is #2  
 = >4 = the number of lines to read is 4  
 = >myString = buffer to get the part of buffer (65536 characters maximum)

index	Data1	Data2	Data3	Data4	Data5
0 →	1	10	0.1	21	100
1 →	2	20	0.2	22	102
2 →	3	30	0.3	23	103
3 →	4	40	0.4	24	104
5 →	5	50	0.5	25	105

“myString” result:

2;20;0.2;22;102  
 3;30;0.3;23;103  
 4;40;0.4;24;104  
 5;50;0.5;25;105

**Prototype**

```
int GatheringDataMultipleLinesGet(
    int SocketID,
    int IndexPoint,
    char * DataBufferLine
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
IndexPoint	int	Index of an acquired data from the current gathering buffer.
NbLines	int	Number of lines to get.

**Output parameters**

DataBufferLine	char *	String contains values from the current buffer at the selected index.
----------------	--------	---

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -32: Gathering not configured.

### 6.2.1.39 GatheringExternalConfigurationGet

#### Name

**GatheringExternalConfigurationGet** – Returns the current configuration of an externally triggered data gathering.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Gathering must be configured: (-32)

#### Description

This function returns the current configuration of an externally triggered data gathering.

Use the “GatheringExternalListGet” function to retrieve a complete list of external gathering types.

For a more thorough description of the external data gathering capability, please refer to section Data Gathering/External Data Gathering of XPS Motion Tutorial.

#### Prototype

```
int GatheringExternalConfigurationGet(int SocketID, char * TypeList)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

TypeList	char *	List of configured gathering types (separator is semicolon).
----------	--------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -32: Gathering not configured.

### **6.2.1.40 GatheringExternalConfigurationSet**

#### **Name**

**GatheringExternalConfigurationSet** – Configures an external data gathering.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input external gathering mnemonic: (-29)
- Gathering must not be in progress: (-43)

#### **Description**

Defines one or several types of data gathered during externally triggered data gathering.

Maximum of 1000000 points can be acquired.

Maximum of 25 data types can be configured in a gathering.

Refer to section 7.4: “External Gathering Data Types”.

The “GatheringExternalListGet” function can be used to retrieve a complete list of gathering types.

For more thorough description of the external data gathering capability, please refer to section Data Gathering/External Data Gathering XPS Motion Tutorial.

#### **Prototype**

```
int GatheringExternalConfigurationSet(
    int SocketID,
    int NbElements,
    char * TypeArray
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NbElements	int	Number of types.
TypeArray	char *	Array of configured gathering types.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -29: Mnemonic gathering type doesn't exist.
- -43: Gathering running.

### 6.2.1.41 GatheringExternalCurrentNumberGet

#### Name

**GatheringExternalCurrentNumberGet** – Returns the current and maximum number of externally gathered data points.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- External gathering must be configured: (-32)

#### Description

This function returns the current and maximum number of data points gathered during an externally triggered data gathering.

For more thorough description of external data gathering capability, please refer to section Data Gathering/External Data Gathering XPS Motion Tutorial.

#### Prototype

```
int GatheringExternalCurrentNumberGet(  
    int SocketID,  
    int * CurrentNumber,  
    int * MaxSamplesNumber  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

CurrentNumber	int *	Current number during acquisition.
MaxSamplesNumber	int *	Maximum number of samples.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -32: Gathering not configured.

### 6.2.1.42 GatheringExternalDataGet

#### Name

**GatheringExternalDataGet** – Reads one line of data from current external gathering buffer.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check index number: (-17)
  - IndexPoint  $\geq 0$ .
  - IndexPoint <currently gathered data number.
- Check gathering state: (-32)

#### Description

This function reads a line of data from current gathering gathering buffer. The buffer line number is defined by the index of an acquired point.

The separator is “;” in the returned data line.

Gathering must be configured in order to use this function, otherwise (-32) error is returned.

#### Prototype

```
int GatheringExternalDataGet(  
    int SocketID,  
    int IndexPoint,  
    char * DataBufferLine  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
IndexPoint	int	Index of an acquired data from the current gathering buffer.

#### Output parameters

DataBufferLine	char *	String contains values from the current buffer at the selected index.
----------------	--------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -32: Gathering not configured.

### 6.2.1.43 GatheringExternalStopAndSave

#### Name

**GatheringExternalStopAndSave** – Stops externally triggered data gathering and saves the data into the XPS controller.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check number of data (>0): (-30)
- Check file opening: (-60)

#### Description

This function stops externally triggered data gathering and saves the data into the XPS controller. Gathered data is stored in the “GatheringExternal.dat” file under “..\\Public” folder of XPS controller.

For more thorough description of external data gathering capability, please refer to section Data Gathering/External Data Gathering of XPS Motion Tutorial.

#### Prototype

```
int GatheringExternalStopAndSave(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	---

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -30: Gathering not started.
- -60: Error during file writing or file doesn't exist.

### 6.2.1.44 GatheringReset

#### Name

**GatheringReset** – Resets gathered data to start new gathering.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Gathering must not be in progress: (-43)

#### Description

This function resets to start a brand new gathering.

The number of gathered data is set to zero.

For more thorough description of internal data gathering capability, please refer to section Data Gathering/Internal Data Gathering of XPS Motion Tutorial.

#### Prototype

```
int GatheringReset(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -43: Gathering running.

### 6.2.1.45 GatheringRun

#### Name

**GatheringRun** – Starts to gather data.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Gathering must be configured: (-32)
- Gathering must not be in progress: (-43)

#### Description

This function starts data gathering.

Data gathering needs to be configured before using this function (See GatheringConfigurationSet function)

The parameters are the number of data points to be gathered and the divisor of the frequency (servo frequency) at which the data gathering will be done.

For more thorough description of the internal data gathering capability, please refer to section Data Gathering/Internal Data Gathering of XPS Motion Tutorial.

#### Prototype

int **GatheringRun**(

```
    int SocketID,  
    int DataNumber,  
    int Divisor  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
DataNumber	int	The number of data line to gather.
Divisor	int	The divisor of the servo frequency.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -32: Gathering not configured.
- -43: Gathering running.

### 6.2.1.46 GatheringRunAppend

#### Name

**GatheringRunAppend** – : Restarts gathering from the point it was stopped.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Gathering must be configured: (-32)
- Gathering must not be in progress: (-43)

#### Description

Restarts gathering from the data point it was stopped as long as gathering current data number has not reached the *DataNumber* previously specified by *GatheringRun()* function. This function repeats the gathering from the data point that was previously stopped, while the gathering current data number has not reached the *DataNumber* previously specified using the *GatheringRun()* function.

The gathering must be configured, executed and stopped before using this function (see *GatheringConfigurationSet*, *GatheringRun*, *GatheringStop* functions)

For more thorough description of the internal data gathering capability, please refer to section Data Gathering/Internal Data Gathering of XPS Motion Tutorial.

#### Prototype

```
int GatheringRunAppend(  
    int SocketID  
);
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -32: Gathering not configured.
- -43: Gathering running.

### 6.2.1.47 GatheringStop

#### Name

**GatheringStop** – Stops internally and externally triggered data gathering.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check number of data (>0): (-30)
- Check file opening: (-60)

#### Description

This function stops internally and externally triggered data gathering. To save to a file, use GatheringStopAndSave function.

For more thorough description of data gathering capability, please refer section Data Gathering/ of XPS Motion Tutorial.

#### Prototype

```
int GatheringStop(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -30: Gathering not started.
- -60: Error during file writing or file doesn't exist.

### 6.2.1.48 GatheringStopAndSave

#### Name

**GatheringStopAndSave** – Stops internally triggered data gathering and saves data into the XPS controller.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check number of data (>0): (-30)
- Check file opening: (-60)

#### Description

This function stops internally triggered data gathering as well as saves the data to the XPS controller. Data is stored in GATHERING.DAT file under “..\\Public” folder of the XPS controller.

For more thorough description of internal data gathering capability, please refer to section Data Gathering/Internal Data Gathering of XPS Motion Tutorial.

#### Prototype

```
int GatheringStopAndSave(
```

```
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	---

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -30: Gathering not started.
- -60: Error during file writing or file doesn't exist.

### 6.2.1.49 GetLibraryVersion

#### Name

**GetLibraryVersion** – Returns the version of the DLL library.

#### Input tests

None.

#### Description

This function returns the version of DLL library.

The library version represents the firmware version that was used to build the library.

#### Prototype

```
int GetLibraryVersion(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

LibVersion	char *	DLL library version.
------------	--------	----------------------

#### Return

None.

### 6.2.1.50 GlobalArrayGet

#### Name

**GlobalArrayGet** – Gets the variable value from the global array.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Verify the index number [0:100[: (-17)

#### Description

This function gets the variable value from the global array, related to “Number” index in a string format.

The first variable value from the global array is referenced to index “0”.

---

#### NOTE

**The number of data points in the global array is limited to 100.**

---

#### Prototype

```
int GlobalArrayGet(  
    int SocketID,  
    int Number,  
    char * StringValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Number	int	Index in the global array.

#### Output parameters

StringValue	char *	Variable value.
-------------	--------	-----------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.51 GlobalArraySet

#### Name

**GlobalArraySet** – Sets the value of the global array.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Verify the index number [0:100[: (-17)

#### Description

This function sets a new value in the global array related to the “Number” index and the new value is set to a string.

---

#### NOTE

**The first variable value of the global array is always referenced to the index “0”.**

**The number of data points in the global array is limited to 100, so the last index is “99”.**

---

#### Prototype

```
int GlobalArraySet(  
    int SocketID,  
    int Number,  
    char * StringValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Number	int	Index in the global array.
StringValue	char *	Variable value.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.52 **GPIOAnalogGainGet**

#### Name

**GPIOAnalogGainGet** – Gets the gain for one or several analog inputs (ADC).

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check board: (-8)
- GPIO name (ADC): (-8)
- Hardware compatibility or XPS initialization in progress: (-22)

#### Description

Gets the gain value for one or several analog inputs. Please refer to Appendix *B.5 Analog I/O* of the XPS Motion Tutorial for further information about ADC gain.

The gain value must be 1, 2, 4 or 8.

The maximum number of INT boards that can be plugged inside the XPS controller is 2, increasing the number of analog inputs (ADC) from 4 to 8.

#### Prototype

```
int GPIOAnalogGainGet(
    int SocketID,
    int NbElements,
    char * GPIONameList,
    double * AnalogGainValueArray
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NbElements	int	Number of analog GPIO to read.
GPIONameList	char *	List of analog input names – separator is comma.

#### Output parameters

AnalogGainValueArray    int \*    Value of analog input gain.

#### Return (In addition to the results of testing the inputs of functions)

- 0:              No error.
- -8:             Wrong object type for this command.
- -22:           Not allowed action.

### 6.2.1.53 **GPIOAnalogGainSet**

#### **Name**

**GPIOAnalogGainSet** – Sets a gain for one or several analog inputs (ADC).

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check board: (-8)
- GPIO name (ADC): (-8)
- Check output value (1, 2, 4 or 8): (-17)
- Hardware compatibility or XPS initialization in progress: (-22)

#### **Description**

Sets a gain value for one or several analog inputs.

The gain value can be 1, 2, 4 or 8

If the conversion of the gain value to bits fails then (-22) error is returned.

The maximum number of INT boards, that can be plugged inside the XPS controller, is 2, increasing the number of analog inputs from 4 to 8.

#### **Prototype**

```
int GPIOAnalogGainSet(
    int SocketID,
    int NbElements,
    char * GPIONameList,
    int * AnalogGainValueArray
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NbElements	int	Number of analog GPIO to read.
GPIONameList	char *	List of analog input names – separator is comma.
AnalogGainValueArray	int *	Value of analog input gain.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.

### 6.2.1.54 GPIOAnalogGet

#### Name

**GPIOAnalogGet** – Reads one or several analog inputs (ADC) or outputs( DAC).

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- GPIO name (ADC or DAC): (-8)
- Hardware compatibility or XPS initialization in progress: (-22)

#### Description

Reads one or several analog IO and returns the value(s) in an array.

#### Prototype

```
int GPIOAnalogGet(  
    int SocketID,  
    int NbElements,  
    char * GPIONameList,  
    double * AnalogValueArray  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NbElements	int	Number of analog GPIO to read.
GPIONameList	char *	List of analog GPIO names – separator is comma.

#### Output parameters

AnalogGainValueArray    double \*    Analog GPIO value array (DAC or ADC).

#### Return (In addition to the results of testing the inputs of functions)

- 0:         No error.
- -8:        Wrong object type for this command.
- -22:      Not allowed action.

### 6.2.1.55 GPIOAnalogRangeConfigurationGet

#### Name

**GPIOAnalogRangeConfigurationGet** – Gets GPIO DAC range configuration for analog output (DAC).

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check if the GPIO type is DAC: (-8)

#### Description

This API returns the DAC range value.

#### Prototype

```
int GPIOAnalogRangeConfigurationGet(  
    int SocketID,  
    char * GPIOName,  
    double * DACRange  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GPIOName	char *	GPIO name.

#### Output parameters

DACRange	double *	DAC range value.
----------	----------	------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.56    GPIOAnalogRangeConfigurationSet**

#### **Name**

**GPIOAnalogRangeConfigurationSet** – Sets GPIO DAC range configuration for analog output(DAC).

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check if the GPIO type is DAC: (-8)

#### **Description**

This API sets the DAC range value. Set values are:

Input value = 0 => Disabled

Input value  $\geq 12.0$  V => DAC range = 12.288 V

Input value  $\in [10.0, 12.0]$  => DAC range = 10 V

Input value  $\in [5.0, 10.0]$  => DAC range = 5 V

#### **Prototype**

```
int GPIOAnalogRangeConfigurationSet(  
    int SocketID,  
    char * GPIOName,  
    double DACRange  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GPIOName	char *	GPIO name.
DACRange	double	DAC range value.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.57 GPIOAnalogSet

#### Name

**GPIOAnalogSet** – Sets one or several analog outputs (DAC).

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check if the GPIO type is DAC: (-8)
- Check input parameters number: (-17)

#### Description

Sets analog value array for DAC type of GPIO.

#### Prototype

```
int GPIOAnalogSet(
```

```
    int SocketID,  
    int NbElements,  
    char * GPIONameList,  
    double * AnalogValueArray  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
NbElements	int	Number of analog GPIO to read.
GPIONameList	char *	List of analog GPIO names – separator is comma.
AnalogGainValueArray	double *	Analog GPIO value array (DAC or ADC).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Not allowed action.
- -22: Not allowed action.

### 6.2.1.58 GPIODigitalGet

#### Name

**GPIODigitalGet** – Reads one digital input or output.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- GPIO name (DI or DO): (-8)
- Hardware compatibility or XPS initialization in progress: (-22)

#### Description

Returns the value of digital input (DI) or digital output (DO).

#### Prototype

```
int GPIODigitalGet(
```

```
    int SocketID,  
    char * GPIOName,  
    unsigned int * DigitalValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GPIOName	char *	Digital GPIO name (maximum size = 250).

#### Output parameters

AnalogGainValueArray	uint *	Analog GPIO value array (DAC or ADC).
----------------------	--------	---------------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -22: Not allowed action.

### 6.2.1.59 GPIODigitalSet

#### Name

**GPIODigitalSet** – Sets one digital output.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- GPIO name (DO): (-8)
- Hardware compatibility or XPS initialization in progress: (-22)

#### Description

Sets the value of the selected digital output (DO).

#### Prototype

```
int GPIODigitalSet(
```

```
    int SocketID,  
    char * GPIOName,  
    unsigned short Mask,  
    unsigned short DigitalOutputValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GPIOName	char *	Digital GPIO name (maximum size = 250).
Mask	ushort	Mask.
DigitalOutputValue	ushort	Digital output value.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -22: Not allowed action.

### 6.2.1.60 **GPIODigitalPulseWidthGet**

#### Name

**GPIODigitalPulseWidthGet** – Reads current GPIO digital I/O pulse width.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check GPIO name and type (must be Digital I/O): (-8)

#### Description

This function reads the current GPIO digital I/O pulse width defined in microseconds.

#### Prototype

```
int GPIODigitalPulseWidthGet(  
    int SocketID,  
    char GPIOName,  
    double * PulseWidth  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the TCP_ConnectToServer” function.
GPIOName	int	GPIO digital I/O name.

#### Output parameters

PulseWidth	double *	Current GPIO pulse width (μsec).
------------	----------	----------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.61 GPIODigitalPulseWidthSet

#### Name

**GPIODigitalPulseWidthSet** – Sets GPIO digital I/O pulse width ( $\mu$ sec).

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check GPIO name and type (must be Digital I/O): (-8)

#### Description

This function configures the GPIO digital I/O pulse width defined in microseconds.

#### Prototype

```
int GPIODigitalPulseWidthSet(  
    int SocketID,  
    char GPIOName,  
    double PulseWidth  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the TCP_ConnectToServer” function.
GPIOName	int	GPIO digital I/O name.
PulseWidth	double	GPIO digital I/O pulse width ( $\mu$ sec).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.62    GroupAccelerationCurrentGet**

#### **Name**

**GroupAccelerationCurrentGet** – Gets the current acceleration for one or all positioners of the selected group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type: (-8)
- Check positioner name: (-18)
- Check group name: (-19)

#### **Description**

Gets the current acceleration for one or all positioners of the selected group.

#### **Prototype**

```
int GroupAccelerationCurrentGet(
    int SocketID,
    char * GroupName[250],
    int NbPositioners,
    double * CurrentAcceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group or positioner name.
NbPositioners	int	Number of positioners in the group.

#### **Output parameters**

CurrentAcceleration	double *	Current acceleration array.
---------------------	----------	-----------------------------

#### **Return** (In addition to the results of testing the inputs of functions)

- 0:              No error.
- -8:             Wrong object type for this command.
- -18:           Positioner name doesn't exist or unknown command.
- -19:           Group name doesn't exist or unknown command.

### **6.2.1.63    GroupAccelerationSetpointGet**

#### **Name**

**GroupAccelerationSetpointGet** – Returns the setpoint acceleration for one or all positioners of the selected group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### **Description**

Returns the setpoint acceleration for one or all positioners of the selected group.

#### **Prototype**

```
int GroupAccelerationSetpointGet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    double * SetpointAcceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.

#### **Output parameters**

SetpointAcceleration      double \*      Setpoint Acceleration array.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0:      No error.
- -8:     Wrong object type for this command.
- -19:    GroupName doesn't exist or unknown command.

### 6.2.1.64 GroupAnalogTrackingModeDisable

#### Name

**GroupAnalogTrackingModeDisable** - Exits the analog tracking mode.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Valid group name: (-19)
- Group status must be "ANALOG TRACKING": (-22)

#### Description

Disables the analog tracking mode.

The group exits the "ANALOG TRACKING" state returning to "READY" state.

If the group state is not "ANALOG TRACKING", (-22) error is returned.

---

#### NOTES

The tracking mode interprets ADC value as a position command or as a velocity command.

To enable the analog tracking mode use "GroupAnalogTrackingModeEnable" function.

---

#### Prototype

```
int GroupAnalogTrackingModeDisable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
GroupName	char *	Group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -9: Wrong number of parameters in the command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.65 GroupAnalogTrackingModeEnable**

#### **Name**

**GroupAnalogTrackingModeEnable** - Enables the analog tracking mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid tracking type (“Position” or “Velocity”): (-8)
- Valid group name: (-19)
- Group status must be “READY”: (-22)
- Configured tracking: (-22)

#### **Description**

Enables the analog tracking mode. To use this function, the group must be in READY state and tracking must be configured before, otherwise an error (-22) is returned.

Once the tracking mode is enabled, the group status must be “ANALOG TRACKING” (48 is the code for Analog tracking state due to a TrackingEnable command).

##### **“Position” analog tracking**

In case of “Position” tracking type, the analog input is interpreted as a position command. The parameters must be set by “AnalogTrackingPositionParametersSet” function and can be read by “AnalogTrackingPositionParametersGet” function.

##### **“Velocity” analog tracking**

In case of “Velocity” tracking type, the analog input is interpreted as a velocity command. The parameters must be set by “AnalogTrackingVelocityParametersSet” function and can be read by “AnalogTrackingVelocityParametersGet” function.

---

#### **NOTE**

**To disable the analog tracking mode use “GroupAnalogTrackingModeDisable” function.**

---

#### **Prototype**

```
int GroupAnalogTrackingModeEnable(
    int SocketID,
    char * GroupName,
    char * Type
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
Type	char *	Tracking type (“Position” or “Velocity”).

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -9: Wrong number of parameters in the command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.66 GroupBrakeStateGet

#### Name

**GroupBrakeStateGet** – Gets current brake command state.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check if GPIO board number: (-17)
- Check if GPIO board is present: (-100)
- Check if “Brake” mode is enabled: (-205)

#### Description

This function reads the current brake command signal state from Inhibit or GPIO connector. Refer to section [BRAKE] in System.ini configuration file. The feature is developed to avoid a run away after hardware error detection (XY group only).

Brake command signal:

- 1) BrakeCommandSignalState parameter is defined “Direct” in System.ini
  - 0 = Brake OFF.
  - 1 = Brake ON.
- 2) BrakeCommandSignalState parameter is defined “Inverted” in System.ini
  - 0 = Brake ON.
  - 1 = Brake OFF.

#### Prototype

```
int GroupBrakeStateGet(  
    int SocketID,  
    char * GroupName,  
    int * BrakeCommand,  
    )
```

#### Input parameters

SocketID	int	Socket identifier gets by the TCP_ConnectToServer” function.
GroupName	char *	Group name (XY).

#### Output parameters

BrakeCommand	int *	Brake command (0 or 1).
--------------	-------	-------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -205: Not enable in your configuration.
- -100: Internal error (memory allocation error ...).

### 6.2.1.67 GroupBrakeSet

#### Name

**GroupBrakeSet** – Set brake command.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check GPIO board number / Check brake command value: (-17)
- Check if GPIO board is present: (-100)
- Check if “Brake” mode is enabled: (-205)

#### Description

This function sets brake command signal from Inhibit or GPIO connector.

Refer to section [BRAKE] in System.ini configuration file. The feature is developed to avoid a run away after hardware error detection (XY group only).

Brake commands:

- 1) BrakeCommandSignalState parameter is defined “Direct” in System.ini
  - 0 = Brake OFF.
  - 1 = Brake ON.
- 2) BrakeCommandSignalState parameter is defined “Inverted” in System.ini
  - 0 = Brake ON.
  - 1 = Brake OFF.

#### Prototype.

```
int GroupBrakeSet(
    int SocketID,
    char *GroupName,
    int BrakeCommand,
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name (XY).
BrakeCommand	int	Brake command (0 or 1).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -205: Not enable in your configuration.
- -100: Internal error (memory allocation error ...).

### 6.2.1.68 GroupCorrectorOutputGet

#### Name

**GroupCorrectorOutputGet** – Returns corrector output for one or all positioners of the selected group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Returns corrector output for one or all positioners of the selected group.

The input parameter “group name” can be a positioner name.

For a group, this function returns the corrector output for each positioner from the selected group.

For a positioner, this function returns only the corrector output associated with the selected positioner.

#### Prototype

```
int GroupCorrectorOutputGet(
```

```
    int SocketID,  
    char * GroupName,  
    int NbPositioners,  
    double * CorrectorOutput  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group (1 if positioner).

#### Output parameters

CorrectorOutput      double \*    Corrector output array.

#### Return (In addition to the results of testing the inputs of functions)

- 0:      No error.

### 6.2.1.69 GroupCurrentFollowingErrorGet

#### Name

**GroupCurrentFollowingErrorGet** – Returns the current following error for one or all positioners of the selected group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### Description

Returns the current following error for one or all positioners of the selected group.

#### Prototype

```
int GroupCurrentFollowingErrorGet(  
    int SocketID,  
    char * GroupName,  
    int NbPositioners,  
    double * CurrentFollowingError  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group (1 if positioner).

#### Output parameters

CurrentFollowingError      double \*    Current following error array.

#### Return (In addition to the results of testing the inputs of functions)

- 0:      No error.
- -18:     Positioner Name doesn't exist or unknown command.
- -19:     GroupName doesn't exist or unknown command.

### 6.2.1.70 GroupGantryModeGet

#### Name

**GroupGantryModeGet** – Gets current gantry option.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the set option mode (Option0, Option1 or Option2): (-17)
- Check the group name is valid (must be XY group): (-19)
- Check gantry mode getting is allowed: (-22)
- Check if XY dual gantry mode is enabled: (-205)

#### Description

Gets the current gantry option. This function is allowed only with a Gantry XY group.

Three “Gantry” options are available:

- **Option0** = >Gantry standard.
- **Option1** = >Gantry force balance.
- **Option2** = >Gantry force balance with dual encoder.

Refer to XY group section from system.ini file to enable “XYDualMode” and to configure it.

#### Prototype

```
int GroupGantryModeGet(  
    int SocketID,  
    char * GroupName,  
    char * Option  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	XY group name.

#### Output parameters

Option	char *	Option0, Option1 or Option2.
--------	--------	------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -205: Not enable in your configuration.

### **6.2.1.71    GroupGantryModeSet**

#### **Name**

**GroupGantryModeSet** – Sets gantry option.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the user option value (Option0, Option1 or Option2): (-17)
- Check the group name (must be XY group): (-19)
- Check if the gantry mode getting is allowed: (-22)
- Check the current gantry option: (-201)
- Check if XY Dual Gantry mode is enabled: (-205)

#### **Description**

Sets the gantry option. It’s possible to configure the gantry option only when the XY group is in “READY” or “DISABLE” state.

Three “Gantry” options are available:

- **Option0** = >Gantry standard.
- **Option1** = >Gantry force balance.
- **Option2** = >Gantry force balance with dual encoder.

Refer to XY group section from system.ini file to enable “XYDualMode” and to configure it.

#### **Prototype**

```
int GroupGantryModeSet(  
    int SocketID,  
    char * GroupName,  
    char * Option  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	XY group name.
Option	char *	Option0, Option1 or Option2.

#### **Output parameters**

None.

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -17: Parameter out of range or incorrect.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -201: The group is already in this mode.
- -205: Not enable in your configuration.

### 6.2.1.72 GroupHomeSearch

#### Name

**GroupHomeSearch** - Initiates a home search.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- The actor must be a valid group name: (-19)
- Group status must be "Not referenced": (-22)

#### Description

This function initiates a home search for each positioner of the selected group.

The group must be initialized and in "NOT REFERENCED" state otherwise error (-22) is returned .

The home search can fail due to:

- a following error: (-25).
- a ZM detection error: (-49).
- a motion done time out, when a dynamic error of the positioner is detected during home search process (-33).
- a home search timeout, when the complete (and complex) home search procedure was not executed in the allowed time: (-28).

For all these errors, the group returns to "NOTINIT" state.

After the home search sequence, each positioner error is checked. If an error is detected, the hardware status register is reset (motor on) and the positioner error is cleared before checking it again. If a positioner error is always present, (-35) error is returned and the group becomes "NOTINIT".

Once the home search is successful, the group is in "READY" state.

---

#### NOTES

**The home search routine for each positioner is defined in "stages.ini" file by "HomeSearchSequenceType" parameter.**

**The homesearch time out is defined in "stages.ini" file by "HomeSearchTimeOut" parameter.**

**The home search sequence is defined in "system.ini" file by "InitializationAndHomeSearchSequence" parameter for each group with several positioners.**

#### XY group

The home search sequence can be "Together", "XthenY" or "YthenX" in a standard XY configuration.

**If the XY group is "Gantry" (dual positioner on X or on Y axis) only "XthenY" or "YthenX" are allowed.**

#### XYZ group

The home search sequence can be "Together" or "XthenYthenZ".

#### MultipleAxes group

The home search sequence can be "Together", "OneAfterAnother" or "OneAfterAnotherInReverseOrder".

If the MultipleAxes group has at least one “Gantry” positioner (dual positioner on one axis or some axes) only “OneAfterAnother” or “OneAfterAnotherInReverseOrder” is allowed.

---

### **Prototype**

```
int GroupHomeSearch(  
    int SocketID,  
    char * GroupName  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -28: Home search timeout.
- -33: Motion done timeout.
- -35: Position is outside of travel limits.
- -49: Inconsistent mechanical zero during home search.

### 6.2.1.73 GroupHomeSearchAndRelativeMove

#### Name

**GroupHomeSearchAndRelativeMove** - Initiates a home search followed by a relative move.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Valid group name: (-19)
- Group status must be "Not referenced": (-22)

#### Description

This function initiates a home search followed by a relative move at the end of the home search.

The group must be initialized and in "NOT REFERENCED" state otherwise (-22) error is returned.

If there is no error, the group status changes to "HOMING".

The home search sequence can fail due to:

- a following error: (-25).
- a ZM detection error: (-49).
- a home search time out: (-33).

For all these errors, the group returns to "NOTINIT" state.

Once the home search is completed, a relative move is executed. After this sequence each positioner is checked for error. If an error is detected, the hardware status register is reset (motor on) and the positioner error is cleared before checking it again. If a positioner error is always present, ERR\_TRAVEL\_LIMITS (-35) is returned and the group state becomes "NOTINIT".

If the home search is successful, the group will be in "READY" state.

---

#### NOTES

The home search routine for each positioner is defined in *stages.ini* file by "HomeSearchSequenceType" parameter.

The home search time out is defined in *stages.ini* file by "HomeSearchTimeOut" parameter.

The home search sequence is defined in *system.ini* file by "InitializationAndHomeSearchSequence" parameter for each group with several positioners:

#### XY group

The home search sequence can be "Together", "XthenY" or "YthenX" if the XY group is standard configuration. If the XY group is Gantry (dual positioner on X or on Y axis) only the "XthenY" or "YthenX" are allowed.

#### XYZ group

The home search sequence can be "Together" or "XthenYthenZ".

#### MultipleAxes group

The home search sequence can be "Together", "OneAfterAnother" or "OneAfterAnotherInReverseOrder".

If the MultipleAxes group has at least one “Gantry” positioner (dual positioner on one axis or some axes), only “OneAfterAnother” or “OneAfterAnotherInReverseOrder” are allowed.

---

### **Prototype**

```
int GroupHomeSearchAndRelativeMove(  
    int SocketID,  
    char * GroupName  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -28: Home search timeout.
- -33: Motion done timeout.
- -35: Position is outside of travel limits.
- -49: Inconsistent mechanical zero during home search.

### 6.2.1.74 GroupInitialize

#### Name

**GroupInitialize** - Initializes the motor and activates the servo loop of the selected group.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Actor must be a group: (-8), (-18)
- Valid group name: (-19)
- Group status must be "NOTINIT": (-22)
- Check state of physical ends of run: (-113)

#### Description

The selected group must be in not initialized "NOTINIT" state, otherwise (-22) error is returned.

This function begins to check the positioner error. If an error is detected, the hardware status register is reset (motor on) and the positioner error is cleared before checking it again. If a positioner error is always present, the motor is turned off, (-5) error is returned and the group state becomes "NOTINIT".

If there is no positioner error, then the group status becomes "MOTOR\_INIT". The master-slave error is cleared, the encoder is reset (update encoder position) and the user travel limits are checked.

If a travel limit error is detected then the motor is turned off, the error (-35) is returned and the group state becomes "NOTINIT".

Moreover, the function checks the state of the physical ends of run. If both physical ends of run are activated, then the motor is turned off, the error (-113) is returned and the group state becomes "NOTINIT".

If no error detected , the motor is initialized in case of "AnalogSinAcc" or "AnalogDualSinAcc". The error (-50) is returned, if the initialization failed and the group state becomes "NOTINIT".

If successful, the positions are reset, the servo loop is activated and the motor is on. The group is now in "NOT REFERENCED" state.

---

#### NOTES

**In Master-Slave mode, after an emergency stop, the master group and the slave group are in "NOTINIT" state.**

**To restart a master-slave relation the slave group(s) must be reinitialized before the master group.**

---

**Prototype**

```
int GroupInitialize(  
    int SocketID,  
    char * GroupName  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -5: Not allowed due to a positioner error or hardware status.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -35: Position is outside of travel limits.
- -50: Motor initialization error. Check InitializationAccelerationLevel, ScalingAcceleration, MaximumJerkTime, EncoderResolution or EncoderScalePitch.
- -113: Both ends of run activated.

### **6.2.1.75 GroupInitializeNoEncoderReset**

#### **Name**

**GroupInitializeNoEncoderReset** - Initializes the motor without encoder reset and activates the servo loop of the selected group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valid group name: (-19)
- Group status must be “NOTINIT”: (-22)
- Check state of physical ends of run: (-113)

#### **Description**

The selected group must be in “NOTINIT” state, otherwise (-22) error is returned.

This function begins to check the positioner error. If an error is detected, the hardware status register is reset (motor on) and the positioner error is cleared before checking it again. If a positioner error is always present, the motor is turned off, (-5) error is returned and the group state becomes “NOTINIT”.

If there is no positioner error, then the group status becomes “MOTOR\_INIT”. The master-slave error is cleared, the encoder is reset (update encoder position) and the user travel limits are checked. If a travel limit error is detected then the motor is turned off, the error (-35) error is returned and the group becomes “NOTINIT”.

Moreover, the function checks the state of the physical ends of run. If both physical ends of run are activated, then the motor is turned off, the error (-113) error is returned and the group state becomes “NOTINIT”.

If no error detected, the motor is initialized in case of “AnalogSinAcc” or “AnalogDualSinAcc”. The error (-50) is returned if the initialization has failed and the group state becomes “NOTINIT”.

If successful, the positions are not reset, the servo loop is activated and the motor is on. The group is now in “NOT REFERENCED” state.

#### **NOTES**

**In Master-Slave mode, after an emergency stop, the master group and the slave group are in “NOTINIT” state.**

**To restart a master-slave relation the slave group(s) must be reinitialized before the master group.**

#### **Prototype**

```
int GroupInitializeNoEncoderReset(
    int SocketID,
    char * GroupName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -5: Not allowed due to a positioner error or hardware status.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -35: Position is outside of travel limits.
- -50: Motor initialization error. Check InitializationAccelerationLevel, ScalingAcceleration, MaximumJerkTime, EncoderResolution or EncoderScalePitch.
- -113: Both ends of run activated.

### 6.2.1.76 GroupInitializeWithEncoderCalibration

#### Name

**GroupInitializeWithEncoderCalibration** - Initializes motor, calibrates encoder and activates servo loop.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Actor must be a group: (-8), (-18)
- Valid group name: (-19)
- Group status must be "NOTINIT": (-22)
- Check state of physical ends of run: (-113)

#### Description

If the selected group is not in "NOTINIT" state, error (-22) is returned

Initializes the motor, calibrates the encoder and activates the servo loop of each positioner of the selected group. To get the calibration results for each positioner, use the "PositionerEncoderCalibrationParametersGet" function.

This function checks the positioner error. If an error is detected, the hardware status register is reset (motor on) and the positioner error is cleared before checking it again. If a positioner error is always present, the motor is turned off, (-5) error is returned and the group state becomes "NOTINIT".

If no positioner error detected , then the group status becomes "MOTOR\_INIT". The master-slave error is cleared, the encoder is reset (update encoder position) and the user travel limits are checked. If a travel limit error is detected then the motor is turned off, the error (-35) is returned and the group state becomes "NOTINIT".

Moreover, the function checks the state of the physical ends of run. If both physical ends of run are activated, then the motor is turned off, the error (-113) is returned and the group state becomes "NOTINIT".

If no error detected, the motor is initialized in case of "AnalogSinAcc" or "AnalogDualSinAcc". The error (-50) is returned if the initialization has failed and the group state becomes "NOTINIT".

After the group initialization the group status is "MOTOR\_INIT", next the encoder undergoes calibration and the group status becomes "ENCODER\_CALIBRATING". If a following error occurs during calibration, (-25) error is returned and the group state becomes "NOTINIT".

If successful, the motor is initialized, the encoder is calibrated and the servo loop is activated. The group is now in "NOT REFERENCED" state.

---

#### NOTE

**In Master-Slave mode, after an emergency stop, the master group and the slave group are in "NOTINIT" status.**

**To restart a master-slave relation the slave group(s) must be reinitialized before the master group.**

---

### **Prototype**

```
int GroupInitializeWithEncoderCalibration(  
    int SocketID,  
    char * GroupName  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -5: Not allowed due to a positioner error or hardware status.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -35: Position is outside of travel limits.
- -50: Motor initialization error. Check InitializationAccelerationLevel, ScalingAcceleration, MaximumJerkTime, EncoderResolution or EncoderScalePitch.
- -113: Both ends of run activated.

### 6.2.1.77 **GroupInterlockDisable**

#### Name

**GroupInterlockDisable** – Disables group interlock mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valid group name: (-19)

#### Description

This function removes the dependency between a group and the groups that are included in GroupInterlock mode. So, when it is executed, the group can initialize, home or move independent of all errors coming from other interlocked groups.

**GroupInterlock mode:** Actions that a group takes based on the activities of other groups: execute actions (like stop axis, power-off, change state...) immediately after an error (or an user command Disable/KillGroup) detected from one of its interlocked groups.

**Example:** The list of interlocked groups is G1, G2, G3, this means:

- G1 depends on G2 and G3 (G1 in action if an error occurs on G2 or G3)
- G2 depends on G1 and G3 (G2 in action if an error occurs on G1 or G3)
- G3 depends on G1 and G2 (G3 in action if an error occurs on G1 or G2)

The interlocked groups are listed in the [GROUPS] section of *system.ini* file:

*InterlockedGroups* = ...; Names of groups involved in the GroupInterlock mode.

The GroupInterlock mode is enabled by default at boot of the XPS controller.

#### Prototype

```
int GroupInterlockDisable(
    int SocketID,
    char *GroupName
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.78 GroupInterlockEnable**

#### **Name**

**GroupInterlockEnable** – Enables group interlock mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valid group name: (-19)

#### **Description**

This function enables the dependency between a group and the groups that are involved in GroupInterlock mode. So, if this function executes, the group cannot initialize, home or move without correcting the errors coming from its interlocked groups.

**GroupInterlock mode:** Activities that a group takes are dependant on the activities of other groups. For example , execute actions like stop axis, power-off, change state immediately after an error or Disable/KillGroup command sent by user detected from one of its interlocked groups.

**Example:** The list of interlocked groups is G1, G2, G3 , this means:

- G1 depends on G2 and G3 (G1 in action if an error occurs on G2 or G3)
- G2 depends on G1 and G3 (G2 in action if an error occurs on G1 or G3)
- G3 depends on G1 and G2 (G3 in action if an error occurs on G1 or G2).

The interlocked groups are listed in the [GROUPS] section of *system.ini* file:

*InterlockedGroups = ...;* Names of groups involved in the GroupInterlock mode.

The GroupInterlock mode is enabled by default at boot of the XPS controller.

#### **Prototype**

```
int GroupInterlockEnable(
    int SocketID,
    char * GroupName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.79 GroupJogCurrentGet

#### Name

**GroupJogCurrentGet** – Returns the current velocity and acceleration from the jog profiler.




---

#### CAUTION

The jog mode cannot be used with a spindle group.

---

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### Description

This function returns the current velocity and acceleration from the jog profiler for one positioner or for all positioners of the selected group.

It must be called when the group is in “JOGGING” mode, otherwise the returned current velocity and current acceleration values will be null.

#### Prototype

```
int GroupJogCurrentGet(
    int SocketID,
    char *GroupName,
    int NbPositioners,
    double *Velocity,
    double *Acceleration
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group (1 if a positioner).

#### Output parameters

Velocity	double *	Current velocity array.
Acceleration	double *	Current Acceleration array.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.80    GroupJogModeDisable**

#### **Name**

**GroupJogModeDisable** – Disables the jog mode \*

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group): (-8)
- Valid group name: (-19)
- Group status must be “JOGGING”: (-22)

#### **Description**

Disables the Jog mode. To use this function, the group must be in “JOGGING” state and all positioners must be idle (i.e velocity equal to 0).

This function exits the “JOGGING” state and returns to “READY” state.

If the group state is not in “JOGGING” state, or the profiler velocity is not null, error (-22) is returned.

#### **NOTE**

To enable the jog mode use “GroupJogModeEnable” function.



#### **CAUTION**

The jog mode cannot be used with spindle group.

#### **Prototype**

```
int GroupJogModeDisable(
    int SocketID,
    char *GroupName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.81 GroupJogModeEnable

#### Name

**GroupJogModeEnable** – Enables the jog mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group): (-8), (-18)
- Valid group name: (-19)
- Group status must be “READY”: (-22)
- Backlash must not be activated: (-46)

#### Description

Enables the Jog mode. To use this function, the group must be in “READY” state and all positioners must be idle (i.e velocity equal to 0).

This function enters “JOGGING” state.

If the group state is not “READY”, (-22) error is returned.

---

#### NOTE

To disable the jog mode use the “GroupJogModeDisable” function.

---



---

#### CAUTION

The jog mode cannot be used with spindle group.

---

#### Prototype

```
int GroupJogModeEnable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -46: Not allowed action due to backlash.

### 6.2.1.82 GroupJogParametersGet

#### Name

**GroupJogParametersGet** – Returns the velocity and acceleration set by “GroupJogParametersSet”.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### Description

This function returns the velocity and acceleration in jog mode set by the user for one positioner or for all positioners of the selected group.

It must be called when the group is in “JOGGING” mode, otherwise both the velocity and the acceleration will be null.

To change the velocity and the acceleration on the fly in jog mode, use “GroupJogParametersSet” function.




---

#### CAUTION

The jog mode cannot be used with spindle group.

---

#### Prototype

```
int GroupJogParametersGet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    double * Velocity,
    double * Acceleration
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.

#### Output parameters

Velocity	double *	User jog velocity array.
----------	----------	--------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.83 GroupJogParametersSet

#### Name

**GroupJogParametersSet** – Changes the velocity and acceleration on the fly, in jog mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)
- Group status must be “JOGGING”: (-22)
- Input parameters for each positioner:
  - Velocity >MaximumVelocity =>Velocity = MaximumVelocity
  - Velocity <-MaximumVelocity =>Velocity = -MaximumVelocity
  - Acceleration ≤0 (-42)
  - Acceleration >MaximumAcceleration => Acceleration = MaximumAcceleration

#### Description

This function changes the velocity and acceleration in jog mode on the fly, If an error occurs, each positioner stops and the velocity value is set to zero.

To use this function, the jog mode must be enabled (requires call of the “GroupJogModeEnable” function).

If the group status is not “JOGGING”, error (-22) is returned.

If a slave or following error are detected during the jog setting, then (-25) or (-44) errors are returned. In this case, the motion is stopped, the jog mode is disabled and the group status becomes “DISABLE”.



#### CAUTION

The jog mode cannot be used with spindle group.

---

#### Prototype

```
int GroupJogParametersSet(  
    int SocketID,  
    char *GroupName,  
    int NbPositioners,  
    double Velocity,  
    double Acceleration  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.
Velocity	double	User jog velocity array.
Acceleration	double	User jog Acceleration array.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -42: Jog value out of range.
- -44: Slave error disabling master.

### 6.2.1.84 GroupKill

#### Name

**GroupKill** - Kills the selected group.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Valid object type (group): (-8), (-18)
- Valid group name: (-19)

#### Description

Kills the selected group to stop its action. The group returns to "NOTINIT" state. If the group is already in this state then it stays there.

The GroupKill is a high priority command that is executed in any condition.

---

#### NOTE

If an initialization, encoder calibration, homing, referencing, motion or trajectory process is in progress, an "emergency stop" will be executed.

Error (-26) will be generated, for each of these functions.

---

#### Prototype

```
int GroupKill(  
    int SocketID,  
    char *GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
GroupName	char *	Group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.85 GroupMotionDisable

#### Name

**GroupMotionDisable** – Disables a “READY” group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valid group name: (-19)
- Group status must be "READY": (-22)

#### Description

Turns OFF the motors, stops the corrector servo loop and disables the position compare mode, if active. The group status becomes “DISABLE”.

If the group is not in “READY” state, error (-22) is returned.

---

#### NOTE

**In the “DISABLED” state the encoder is still read.**

**To return to “READY” state, call “GroupMotionEnable” function.**

---

#### Prototype

```
int GroupMotionDisable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.86 GroupMotionEnable

#### Name

**GroupMotionEnable** – Enables a group in DISABLE state to turn the motors on and to restart corrector loops.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valid group name: (-19)
- Group status must be “DISABLE”: (-22)

#### Description

Turns ON the motors and restarts the corrector servo loops. The group state becomes “READY”.

If the group is not in “DISABLE” error (-22) is returned.

#### Prototype

```
int GroupMotionEnable(  
    int SocketID,  
    char *GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.87 GroupMotionStatusGet**

#### **Name**

**GroupMotionStatusGet** – Returns the motion status for one or all positioners of the selected group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### **Description**

Returns the motion status for one or all positioners of the selected group.

The motion status possible values are:

0: Not moving state (group status in NOT\_INIT, NOT\_REF or READY).

1: Busy state (positioner in moving, homing, referencing, spinning, analog tracking, trajectory, encoder calibrating, slave mode).

#### **Prototype**

```
int GroupMotionStatusGet(
    int SocketID,
    char *GroupName,
    int NbPositioners,
    int *Status
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.

#### **Output parameters**

Status	int *	Positioner status.
--------	-------	--------------------

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.88 GroupMoveAbort

#### Name

**GroupMoveAbort** – aborts the motion or the jog in progress for a group or a positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)
- Group status must be “MOVING” or “JOGGING”: (-22)

#### Description

This function aborts a motion or a jog in progress. The group state must be “MOVING” or “JOGGING”, otherwise error (-22) is returned.

##### **For a group:**

If the group status is “MOVING”, this function stops all motion in progress.

If the group status is “JOGGING”, this function stops all “jog” motions in progress and disables the jog mode. After this “group move abort” action, the group status becomes “READY”.

##### **For a positioner:**

If the group status is “MOVING”, this function stops the motion of the selected positioner.

If the group status is “JOGGING”, this function stops the “jog” motion of the selected positioner.

If the positioner is idle, an error (-22) is returned.

After “positioner move abort” action, if all positioners are idle, the group status changes to “READY”, otherwise it stays in the same state.

#### **NOTE**

**If the “move abort” action fails, error (-27) is returned.**

**This error is generated when GroupMoveAbort is used to abort a motion of a positioner in a group and the name of the positioner is incorrect.**

**This error will also be returned by “GroupMove” function.**

#### Prototype

```
int GroupMoveAbort(
    int SocketID,
    char *GroupName
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -27: Move Aborted.

### 6.2.1.89 GroupMoveAbortFast

#### Name

**GroupMoveAbortFast** – aborts with user-defined deceleration a motion or a jog in progress for a group or positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid UserDecelerationMultiplier value (**≥1** and **≤100**): (-17)
- Valid positioner name: (-18)
- Valid group name: (-19)
- Group status must be “MOVING” or “JOGGING”: (-22)

#### Description

This function aborts a motion or a jog in progress with a deceleration value defined by user (*UserDeceleration*):

*UserDeceleration* = *DecelerationMultiplier* \* *MaximumAcceleration*.

Here: *DecelerationMultiplier*: *GroupMoveAbortFast* function parameter

*MaximumAcceleration*: Stage parameter, defined in the stages.ini file.

The group state must be “MOVING” or “JOGGING”, otherwise error (-22) is returned.

#### **For a group:**

If the group state is “MOVING”, this function stops all motion.

If the group state is “JOGGING”, this function stops all “jog” motion and disables the jog mode. After this “group move abort” action, the group state becomes “READY”.

#### **For a positioner:**

If the group state is “MOVING”, this function stops the motion of the selected positioner.

If the group state is “JOGGING”, this function stops the “jog” motion of the selected positioner.

If the positioner is idle, error (-22) is returned.

After “positioner move abort” action, if all positioners are idle, the group state becomes “READY”, otherwise it stays the same

#### **NOTE**

**If the “move abort” action fails, error (-27) is returned.**

#### Prototype

```
int GroupMoveAbortFast(
    int SocketID,
    char *GroupName
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
DecelerationMultiplier	int	Braking deceleration multiplier.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -27: Move Aborted.

### 6.2.1.90 GroupMoveAbsolute

#### Name

**GroupMoveAbsolute** - Initiates an absolute move for a positioner or a group.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Valid object type (group or positioner): (-8)
- Verify target position in relation with the travel limits: (-17)
  - TargetPosition  $\geq$  MinimumTargetPosition.
  - TargetPosition  $\leq$  MaximumTargetPosition.
- Valid positioner name: (-18)
- Valid group name: (-19)
- Group status must be "READY" or "MOVING": (-22)

#### Description

This function initiates an absolute move to one or all positioners of the selected group. The group state must be "READY" or "MOVING", otherwise error (-22) is returned. If the group is "READY" then the group state becomes "MOVING".

An absolute motion is defined by the distance between the zero position and the target position. If the current position is the same as the target position then no move will be done.

Each "positioner" move refers to the acceleration, velocity, minimum jerkTime and maximum jerkTime as defined in the "Stages.ini" file or as redefined by the "PositionerSGammaParametersSet" function.

If a slave or following error is detected during the move then (-25) or (-44) errors are returned. In this case, the motion in progress is stopped and the group state becomes "DISABLE".

If the "MotionDoneMode" is defined as "VelocityAndPositionWindowMotionDone" error (-33) will be returned when the time out (defined by "MotionDoneTimeout" in the stages.ini file) is reached before the motion is done. The group state will change to "DISABLE".

In case of "GroupMoveAbort", error (-27) is returned., The motion in progress is stopped and the group state becomes "READY".

During a move with PositionCompare (or TimeFlasher) scan enabled, if the current following error exceeds *WarningFollowingError* value inside the PositionCompare (or TimeFlasher) scan zone, a *WarningFollowingErrorFlag* is latched. In this case the motion continues and finishes normally (the group status becomes "READY"), but the *GroupMoveAbsolute* function returns (-120) error instead of SUCCESS (0). To reset *WarningFollowingErrorFlag* for next moves, execute *PositionerPositionCompareDisable*(or *PositionerTimeFlasherDisable*) function.

If in "GroupKill" command an emergency brake or an emergency stop has occurred, error (-26) is returned. In this case, the motion in progress is stopped and the group state becomes "NOTINIT".

---

#### NOTE

**Asynchronous moves for positioners of the same group are possible through the use of different sockets to send functions.**

---

**Prototype**

```
int GroupMoveAbsolute(
    int SocketID,
    char *GroupName,
    int NbPositioners,
    double *TargetPosition
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.
TargetPosition	double *	Target position array.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -26: Kill command or Emergency signal: check each positioners and each slave positioners, check that motion does not exceed software limits when combined with mapping and other features.
- -27: Move Aborted.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -120: Warning following error during move with position compare enabled.

### 6.2.1.91 GroupMoveEndWait

#### Name

**GroupMoveEndWait**– Wait for the end of move (XY group only).

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group is “XY”: (-8)
- Check expected position after motion: (-211)

#### Description

This function allows waiting for the true end of the move (after a GroupMoveAbsolute, GroupMoveRelative or GroupMoveSlice).

It is only available for an XY group.

#### Prototype

```
int GroupMoveEndWait(  
    int SocketID,  
    char *GroupName,  
    double TimeOutMs,  
    double XPosition,  
    double YPosition  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	XY Group name.
TimeOutMs	double	Time out in milliseconds.
XPosition	double	X position to check in controller unit.
YPosition	double	Y position to check in controller unit.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -211: Not expected position after motion.

### 6.2.1.92 GroupMoveRelative

#### Name

**GroupMoveRelative** - Initiates a relative move for a positioner or a group.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Valid object type (group or positioner): (-8)
- Verify target displacement in relation with the travel limits: (-17)
  - TargetPosition  $\geq$  MinimumTargetPosition.
  - TargetPosition  $\leq$  MaximumTargetPosition.
- Valid positioner name: ERR\_POSITIONER\_NAME (-18)
- Valid group name: ERR\_GROUP\_NAME (-19)
- Group status must be "READY" or "MOVING": (-22)

#### Description

This function initiates a relative move defined by the target displacement to one or all positioners of the selected group. The group state must be "READY" or "MOVING". otherwise error (-22) is returned. If the group is in "READY" state , then it turns into "MOVING".

The target displacement and the current position define the new target position to reach:

$$\text{NewTargetPosition} = \text{CurrentTargetPosition} + \text{TargetDisplacement}$$

Each "positioner" move refers to the acceleration, velocity, minimum jerkTime and maximum jerkTime as defined in the "Stages.ini" file or as redefined by the "PositionerSGammaParametersSet" function.

If a slave or following error is detected during the move then errors (-25) or (-44) are returned. In this case, the motion in progress is stopped and the group status becomes "DISABLE".

If "MotionDoneMode" is defined as "VelocityAndPositionWindowMotionDone", error (-33) is returned when the time out (defined by "MotionDoneTimeout" in the stages.ini file) is reached before the motion is done. The group state becomes "DISABLE".

In case "GroupMoveAbort" is executed , error (-27) is returned. The motion in progress is stopped and the group state becomes "READY".

During move with PositionCompare (or TimeFlasher) scan enabled, if the current following error exceeds *WarningFollowingError* value inside the PositionCompare (or TimeFlasher) scan zone, a *WarningFollowingErrorFlag* is latched. In this case the motion continues and ends normally (the group status becomes "READY"), but the *GroupMoveRelative* function returns (-120) error instead of SUCCESS (0). To reset *WarningFollowingErrorFlag* for the next moves, execute *PositionerPositionCompareDisable*(or *PositionerTimeFlasherDisable*) function.

If in "GroupKill" command an emergency brake or an emergency stop has occurred, error (-26) is returned. In this case, the motion in progress is stopped and the group state becomes "NOTINIT".

---

#### NOTE

**Asynchronous moves for positioners of a same group are possible through the use of different sockets to send the functions.**

### **Prototype**

```
int GroupMoveRelative(  
    int SocketID,  
    char * GroupName,  
    int NbPositioners,  
    double * Displacement  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.
Displacement	double *	Relative displacement array.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -26: Kill command or Emergency signal: check each positioners and each slave positioners, check that motion does not exceed software limits when combined with mapping and other features.
- -27: Move Aborted.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -120: Warning following error during move with position compare enabled.

### **6.2.1.93    GroupPositionCurrentGet**

#### **Name**

**GroupPositionCurrentGet** – Returns the current position for one or all positioners of the selected group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid positioner name: (-18)
- Valid group name: (-19)

#### **Description**

Returns the current position for one or all positioners of the selected group.

The current position is defined as:

$$\text{CurrentPosition} = \text{SetpointPosition} - \text{FollowingError}$$

#### **Prototype**

```
int GroupPositionCurrentGet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    double * CurrentPosition
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.

#### **Output parameters**

CurrentPosition	double *	Current position array.
-----------------	----------	-------------------------

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.94    GroupPositionSetpointGet**

#### **Name**

**GroupPositionSetpointGet** – Returns the setpoint position for one or all positioners of the selected group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### **Description**

Returns the setpoint position for one or all positioners of the selected group.

The “setpoint” position is calculated by the motion profiler and represents the “theoretical” position to reach.

#### **Prototype**

```
int GroupPositionSetpointGet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    double * SetpointPosition
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.

#### **Output parameters**

SetpointPosition              double \* Setpoint position array.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0:        No error.
- -8:       Wrong object type for this command.
- -18:      Positioner Name doesn't exist or unknown command.
- -19:      GroupName doesn't exist or unknown command.

### **6.2.1.95 GroupPositionTargetGet**

#### **Name**

**GroupPositionTargetGet** – Returns the target position for one or all positioners of the selected group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### **Description**

Returns the target position for one or all positioners of the selected group.

The target position represents the “end” position after the move.

For instance, during a move from 0 to 10 units, the position values are:

GroupPositionTargetGet = >**10.0000**

GroupPositionCurrentGet = >**5.0005**

GroupPositionSetpointGet = >**5.0055**

#### **Prototype**

```
int GroupPositionTargetGet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    double * TargetPosition
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.

#### **Output parameters**

TargetPosition	double *	Target position array.
----------------	----------	------------------------

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.96 GroupReferencingActionExecute**

#### **Name**

**GroupReferencingActionExecute** – Initiates the given action, with the given sensor and parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid action name and sensor name: (-8)
- Input parameter coherence: (-17)
  - For a “LatchOnHighToLowTransition” or “LatchOnLowToHighTransition” or “LatchOnIndex” or “LatchOnIndexAfterSensorHighToLowTransition” or “MoveToPreviouslyLatchedPosition” action.
  - Parameter  $\leq$ MaximumVelocity.
  - Parameter  $\neq$  0.
  - Referencing state: (-22)
  - Latch must be done since referencing start for a “MoveToPreviouslyLatchedPosition” action.
- Valid positioner name: (-18)
- Group status must be “NOT REFERENCED”: (-22)

#### **Description**

Initiates a referencing action for a positioner. A referencing action is defined by a given action name (see “Action list” below), with a given sensor name (see “Sensor list” below) and parameters. For more detail, see XPS User’s manual referencing section.

<b>Action list</b>	<b>Sensor to be defined</b>
LatchOnHighToLowTransition	Yes
LatchOnIndex	None
LatchOnIndexAfterSensorHighToLowTransition	Yes
LatchOnLowToHighTransition	Yes
MoveRelative	None
MoveToPreviouslyLatchedPosition	None
SetPosition	None
SetPositionToHomePreset	None

#### **Sensor list**

MechanicalZero  
MinusEndOfRun  
PlusEndOfRun  
None.

If a following error occurs during the referencing and motion is in progress, an emergency brake is applied and error (-25) is returned. The group state becomes “NOTINIT”.

If the home search time out is reached, error (-28) is returned. The group state becomes “NOTINIT”.

After referencing is done, to exit the “REFERENCING” state and to go to “READY” state use “GroupReferencingStop” function.



---

**CAUTION**

This function must be used with a positioner.

---

**Prototype**

```
int GroupReferencingActionExecute(  
    int SocketID,  
    char * GroupName,  
    char * Action,  
    char * Sensor,  
    double Parameter  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
Action	char *	Referencing action name.
Sensor	char *	Referencing sensor name.
Parameter	double	Referencing parameter (related to the referencing action).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -28: Home search timeout.

### 6.2.1.97 GroupReferencingStart

#### Name

**GroupReferencingStart** – Starts the referencing mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid group name: (-19)
- Group status must be “NOT REFERENCED”: (-22)

#### Description

Starts the referencing mode and sets the group status to “REFERENCING”.

To use this function, the selected group must be in “NOT REFERENCED” state, or error (22) is returned.

To stop the referencing mode and go to “READY” state, use “GroupReferencingStop” function.

#### Prototype

```
int GroupReferencingStart(
```

```
    int SocketID,  
    char *GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.98 GroupReferencingStop

#### Name

**GroupReferencingStop** – Stops the referencing mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid group name: (-19)
- Group status must be “REFERENCING”: (-22)

#### Description

Stops the referencing mode and sets the group state to “READY”.

To use this function, the selected group must be in “REFERENCING” state, otherwise error (-22) is returned.

The travel limits are checked before stopping referencing mode. Error (-35) is returned, if the profiler position is out of range of the software travel limits and the group stays in the “REFERENCING” state.

#### Prototype

```
int GroupReferencingStop(  
    int SocketID,  
    char *GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -35: Position is outside of travel limits.

### 6.2.1.99 GroupSpinCurrentGet

#### Name

**GroupSpinCurrentGet** – Returns the spin mode parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a Spindle group): (-8)
- Check the positioner name: (-18)

#### Description

This function returns the current (or actual) velocity and acceleration used by the SPIN mode.

#### Prototype

```
int GroupSpinCurrentGet(  
    int SocketID,  
    char * GroupName,  
    double * Velocity,  
    double * Acceleration  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Spindle group name.

#### Output parameters

Velocity	double *	Velocity (units/s).
Acceleration	double *	Acceleration (units/s <sup>2</sup> ).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.100 GroupSpinModeStop

#### Name

**GroupSpinModeStop** – Stops the motion of the spindle group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a Spindle group): (-8)
- Check the positioner name (must be a group name): (-18)

#### Description

This function stops the motion of a spindle group and sets the group state to READY.

To use this function, the group must be in SPINNING state, otherwise error (-22) is returned.

#### Prototype

```
int GroupSpinModeStop(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Spindle group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.101 GroupSpinParametersGet

#### Name

**GroupSpinParametersGet** – Returns the spin profiler parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a spindle group): (-8)
- Check the function (must be a spindle function): (-18)

#### Description

This function returns the “Setpoint” (theoretical) velocity and acceleration used in SPIN mode.

#### Prototype

```
int GroupSpinParametersGet(  
    int SocketID,  
    char * GroupName,  
    double * Velocity,  
    double * Acceleration  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Spindle group name.

#### Output parameters

Velocity	double *	Setpoint Velocity (units/s).
Acceleration	double *	Setpoint Acceleration (units/s <sup>2</sup> ).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.102 GroupSpinParametersSet**

#### **Name**

**GroupSpinParametersSet** – Sets the spin profiler parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a spindle group): (-8)
- Check input parameter value: (-17)
  - Velocity  $\leq$  MaximumVelocity.
  - Velocity  $\geq$  - MaximumVelocity.
  - Acceleration  $> 0$ .
  - Acceleration  $\leq$  MaximumAcceleration.
- Check the function (must be a spindle function): (-18)

#### **Description**

This function starts the SPIN mode and allows on-the-fly changes to the velocity and acceleration used by this mode. If an error occurs, the positioner stops and the velocity value is set to zero.

After the tests on input values:

If Velocity  $>$  MaximumVelocity  $=>$  Velocity = MaximumVelocity  
 If Velocity  $<-$  MaximumVelocity  $=>$  Velocity = - MaximumVelocity  
 If Acceleration  $\leq 0$   $=>$  ERROR and stop motion  
 If Acceleration  $>$  MaximumAcceleration  $=>$  Acceleration = MaximumAcc.

#### **Prototype**

```
int GroupSpinParametersSet(
    int SocketID,
    char * GroupName,
    double Velocity,
    double Acceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Spindle group name.
Velocity	double	Setpoint Velocity (units/s).
Acceleration	double	Setpoint Acceleration (units/s <sup>2</sup> ).

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.103 GroupStatusGet

#### Name

**GroupStatusGet** – Returns the group status code.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid group name: (-19)

#### Description

Returns the group status code. The group status codes are listed in section 7.8: “Group Status List”.

The description of group status code can be retrieved from “GroupStatusStringGet” function.

#### Prototype

```
int GroupStatusGet(  
    int SocketID,  
    char *GroupName,  
    int *GroupStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

GroupStatus	int *	Status of the group.
-------------	-------	----------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.104 GroupStatusStringGet

#### Name

**GroupStatusStringGet** – Gets the group status description from a group status code.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the group status description corresponding to a group status code (see section 7.10: “Controller Status List”).

If the group status code is not referenced, “Error: ( “ undefined status ” ) will be returned.

#### Prototype

```
int GroupStatusStringGet(  
    int SocketID,  
    int GroupStatusCode,  
    char * GroupStatusString  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupStatusCode	int	Group status code.

#### Output parameters

GroupStatusString	char *	Group status description.
-------------------	--------	---------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.105 GroupVelocityCurrentGet

#### Name

**GroupVelocityCurrentGet** – Returns the current velocity for one or all positioners of the selected group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### Description

Returns the current velocity for one or all positioners of the selected group.

#### Prototype

```
int GroupVelocityCurrentGet(  
    int SocketID,  
    char * GroupName,  
    int NbPositioners,  
    double * CurrentVelocity  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
NbPositioners	int	Number of positioners in the selected group.

#### Output parameters

CurrentVelocity	double *	Current Velocity array.
-----------------	----------	-------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.106 GroupVelocitySetpointGet

#### Name

**GroupVelocitySetpointGet** – Gets the setpoint velocity for one or all positioners of the selected group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type: (-8)
- Check positioner name: (-18)
- Check group name: (-19)

#### Description

Gets the setpoint velocity for one or all positioners of the selected group.

#### Prototype

```
int GroupVelocitySetpointGet(  
    int SocketID,  
    char * GroupName[250],  
    int NbPositioners,  
    double * SetpointVelocity  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group or positioner name.
NbPositioners	int	Number of positioners in the group.

#### Output parameters

SetpointVelocity	double *	Setpoint velocity array.
------------------	----------	--------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner name doesn't exist or unknown command.
- -19: Group name doesn't exist or unknown command.

### 6.2.1.107 HardwareDateAndTimeGet

#### Name

**HardwareDateAndTimeGet** – Returns the current date and time.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the current date and time of XPS controller with the format “WeekDay Month Day Hour:Minute:Second Year”, for example “Tue Jan 15 10:28:06 2008”.

---

#### NOTE

**The date and time returned by the controller are not guaranteed and should not be used as a reference.**

---

#### Prototype

```
int HardwareDateAndTimeGet(  
    int SocketID,  
    char * DateAndTime  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

DateAndTime	char *	Controller date and time.
-------------	--------	---------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.108 HardwareDateAndTimeSet

#### Name

**HardwareDateAndTimeSet** – Sets the date and time.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function sets the date and time of the XPS controller. The date format must be “WeekDay Month Day Hour:Minute:Second Year”, for example “Tue Jan 15 10:28:06 2008”.

#### Prototype

```
int HardwareDateAndTimeSet(  
    int SocketID,  
    char * DateAndTime  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
DateAndTime	char *	Controller date and time.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.109 HardwareDriverAndStageGet

#### Name

**HardwareDriverAndStageGet** – Gets smart hardware.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check plug number: (-17)
- Check if the group is not in NOTINIT state: (-22)

#### Description

This function reads the driver reference and the smart stage name (only for a smart stage) from board EEPROMs for the selected positioner plug number.

#### Prototype

```
int HardwareDriverAndStageGet(  
    int SocketID,  
    int PlugNumber,  
    char * DriverName,  
    char * StageName,  
    )
```

#### Input parameters

SocketID	int	Socket identifier gets by the TCP_ConnectToServer” function.
PlugNumber	int	Positioner plug number.

#### Output parameters

DriverName	char *	Driver reference.
StageName	char *	Smart stage name.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.

### 6.2.1.110 InstallerVersionGet

#### Name

**InstallerVersionGet**– Gets the installer pack version installed in the controller.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the installer pack version used for the current firmware installation.

#### Prototype

```
int InstallerVersionGet(  
    int SocketID,  
    char * Version  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Version	char *	Installer pack version.
---------	--------	-------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.111 INTServitudesCommandGet

#### Name

**INTServitudesCommandGet** – Reads INT servitudes command.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the INT servitudes command register value.

#### Prototype

```
int INTServitudesCommandGet(  
    int SocketID,  
    short * INTServitudesCommand  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

INTServitudesCommand	short *	INT servitudes command.
----------------------	---------	-------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -126: Wrong parameter type in the command string: short or short \* expected.

### 6.2.1.112 INTServitudesStatusGet

#### Name

**INTServitudesStatusGet** – Reads INT servitudes status.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the INT servitudes status register value.

#### Prototype

```
int INTServitudesStatusGet(  
    int SocketID,  
    short * INTServitudesStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

INTServitudesStatus	short *	INT servitudes status.
---------------------	---------	------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -126: Wrong parameter type in the command string: short or short \* expected.

### 6.2.1.113 KillAll

#### Name

**KillAll** – Kills all groups.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function kills and resets all groups as well as all analog and digital I/O’s  
The following sequence of steps is performed by KillAll function.

- 1) An “emergency stop” is executed if the group state is defined as:

HOMING  
REFERENCING  
MOVING  
JOGGING  
ANALOG\_TRACKING

- 2) The motor is turned off, the motion done is stopped and the control loop is stopped.
- 3) “ERR\_EMERGENCY\_SIGNAL” is returned by each function in progress, and for groups that are in following states:

MOTOR\_INIT  
ENCODER\_CALIBRATING  
HOMING  
REFERENCING  
MOVING  
TRAJECTORY  
ERR\_EMERGENCY\_SIGNAL

- 4) the group state is not initialized “NOTINIT” for all groups.

#### Prototype

```
int KillAll(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.114 Login

#### Name

**Login** – Self-identification.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the user name and the password: (-106)

#### Description

This function allows a user to identify himself as “SuperUser”, “Administrator” or “User”.

The user account must be exited, otherwise error (-106) is returned.

This function is not meant to be used from the “terminal” web page.

---

#### NOTE

To add a new user account, you must use the XPS web site with “Administrator” rights. In the main menu, select “Controller ” and go to the “Users management” page.

---

#### Prototype

```
int Login(  
    char * Name,  
    char * Password  
)
```

#### Input parameters

Name	char *	User name.
Password	char *	User password.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -106: Wrong user name or password.
- -123: Action not allowed, an Administrator is already logged in.

### **6.2.1.115 `MultipleAxesPTExecution`**

#### **Name**

**MultipleAxesPTExecution** – Executes a PT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length: (-3)
- Check group type (must be a MultipleAxes group): (-8)
- Check input value (number of executions must >0): (-17)
- Check group name: (-19)
- Group state must be "READY": (-22)
- Check backlash (must not be enabled): (-46)
- Check BaseVelocity (stages.ini, must = 0): (-48)
- Check trajectory file existence or file reading: (-61)
- Check message queue: (-71)

#### **Description**

This function executes a PT (Position Time) trajectory. The trajectory file must be stored in “\Admin\Public\Trajectory” folder of the XPS controller. If the trajectory cannot be initialized (message queue or task error), error (-72) is returned.

Before a trajectory execution, it is recommended to check whether the trajectory is within the positioner motion capabilities by using “MultipleAxesPTVerification” and “MultipleAxesPTVerificationResultGet” functions.

During the trajectory execution, if a positioner reaches one of travel limits, the trajectory execution will stop and error (-25) is generated under positioner errors.

#### **NOTE**

In case of errors (-33), (-25) and (-44), the group state changes to DISABLE. To help determine the error source, check the positioner errors, the hardware status and the driver status.

#### **Prototype**

```
int MultipleAxesPTExecution(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250],  
    int ExecutionNumber  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
ExecutionNumber	int	Number of trajectory executions.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -46: Not allowed action due to backlash.
- -48: BaseVelocity must be null.
- -61: Error file corrupt or file doesn't exist.
- -71: Error read from or write in message queue.
- -72: Error trajectory initialization.

### **6.2.1.116 `MultipleAxesPTLoadToMemory`**

#### **Name**

**MultipleAxesPTLoadToMemory** – Loads some lines of PT trajectory to the controller memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function loads some lines of PT trajectory into XPS controller memory. Each trajectory element must be separated by a comma. The trajectory lines are separated between each other by “\n” (LF) character. To verify or to execute the PT trajectory loaded in memory, use “**FromMemory**” string instead of a file name.

#### **NOTES**

- All of the PT functions, when called with the string “FromMemory” instead of a FileName, will perform the same operation as the PVT trajectory in RAM as it does from a disk.

#### **Example:**

*MultipleAxesPTVerification(socketId,myGroup,FromMemory)  
MultipleAxesPTExecution(socketId,myGroup,FromMemory,1).*

#### **Prototype**

```
int MultipleAxesPTLoadToMemory(
    int SocketID,
    char GroupName[250],
    char TrajectoryData[400]
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
TrajectoryData	char *	Trajectory data lines.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### **6.2.1.117 `MultipleAxesPTParametersGet`**

#### **Name**

**MultipleAxesPTParametersGet** – Gets PT trajectory parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Check current executing trajectory type (must be PT): (-22)

#### **Description**

This function returns the PT trajectory parameters (trajectory name and current executing element number) of the current PT trajectory.

#### **Prototype**

```
int MultipleAxesPTParametersGet(  
    int SocketID,  
    char GroupName[250],  
    char * FileName,  
    int * CurrentElementNumber  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

FileName	char *	Currently executing trajectory file name.
CurrentElementNumber	int *	Currently executing element number.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.118 `MultipleAxesPTPulseOutputGet`**

#### **Name**

**MultipleAxesPTPulseOutputGet** – Gets the configuration of pulse generation for PT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)

#### **Description**

This function returns the last configuration of pulse generation of a PT trajectory, that was previously set by *MultipleAxesPTPulseOutputSet()*.

The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

#### ***Example:***

*MultipleAxesPTPulseOutputSet(MyGroup, 3, 5, 0.01)*

*MultipleAxesPTPulseOutputGet(MyGroup) => 0,3,5,0.01 (0 is the error return, meaning “ no error ”)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

Start element= 3

End element = 5

Time interval = 0.01 seconds.

#### **Prototype**

```
int MultipleAxesPTPulseOutputGet(  
    int SocketID,  
    char GroupName[250],  
    int * StartElement,  
    int * EndElement,  
    double * TimeInterval  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

StartElement	int *	Start pulse element number.
EndElement	int *	End pulse element number.
TimeInterval	double *	Time interval between pulses (seconds).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.

### **6.2.1.119 `MultipleAxesPTPulseOutputSet`**

#### **Name**

**MultipleAxesPTPulseOutputSet** – Sets the configuration of pulse generation for PT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Check the pulse generation must not be in progress: (-22)

#### **Description**

This function configures and activates the pulse generation of a PT trajectory. The pulse generation is defined by a start element, an end element, and a time interval in seconds. If a pulse generation is already activated on the selected PT trajectory then this function returns error -22 (“Not allowed action”).

Please note, that the pulse output settings are automatically removed, when the trajectory is over. Hence, with the execution of every new trajectory, it is required to define the pulse output settings again.

This capability allows output of pulses at constant time intervals on a PT trajectory. The pulses are generated between the first and the last trajectory element. The minimum possible time interval is CorrectorISRPeriod value (*system.ref*).

The trajectory pulses are generated on the following GPIO outputs:

<b>GPIO signals</b>	<b>ISA XPS controller</b>	<b>PCI XPS controller (for example XPS-RL) with basic GPIO board</b>	<b>PCI XPS controller (for example XPS-RL) with extended GPIO board</b>
Window	GPIO2, pin 11	GPIO1.DO6	GPIO5.DO14
Pulses	GPIO2, pin 12	GPIO1.DO7	GPIO5.DO15

To find the GPIO connector pin number from GPIOx.DOy, refer to Appendix / General I/O Description of XPS User’s Manual.,

#### ***Example:***

*MultipleAxesPTPulseOutputSet(Group1, 3, 5, 0.01)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

#### **Prototype**

```
int MultipleAxesPTPulseOutputSet(
    int SocketID,
    char GroupName[250],
    int StartElement,
    int EndElement,
    double TimeInterval
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
StartElement	int	Start pulse element number.
EndElement	int	End pulse element number.
TimeInterval	double	Time interval between pulses (seconds).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.120 MultipleAxesPTResetInMemory

#### Name

**MultipleAxesPTResetInMemory** – Deletes the content of PT trajectory buffer in the controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function deletes the PT trajectory buffer, that was previously loaded with “MultipleAxesPTLoadToMemory” function, in the controller memory.

#### Prototype

```
int MultipleAxesPTResetInMemory(  
    int SocketID,  
    char GroupName[250]  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.121 MultipleAxesPTVerification

#### Name

**MultipleAxesPTVerification** – Checks the PT trajectory data file.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length (must  $\leq 250$ ): (-3)
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Check BaseVelocity value (must = 0): (-48)
- Check trajectory file existence and the file format: (-61)
- Check trajectory (number of elements must  $> 0$ ): (-66)
- Check velocity (Minimum Velocity  $\leq$  Velocity  $\leq$  Maximum Velocity): (-68)
- Check acceleration (Minimum acc.  $\leq$  acceleration  $\leq$  Maximum acc.): (-69)
- Check end output velocity (must = 0): (-70)
- Check delta time (DeltaTime must  $> 0$ ): (-75)

#### Description

This function verifies the execution of a PT trajectory. The results of the verification can be obtained by “MultipleAxesPTVerificationResultGet” function. The trajectory file must be stored in “\ADMIN\Public\Trajectory” folder of XPS controller. If the trajectory cannot be initialized (task error), error (-72) is returned.

This function can be executed at any time and is independent of the trajectory execution. It performs the following:

- Checks the trajectory file for data coherence.
- Calculates the trajectory limits,:1) the required travel per positioner, 2)the maximum possible trajectory velocity and 3) the maximum possible trajectory acceleration. This function helps to define the parameters for the trajectory execution.
- The required travel values (MinimumPosition and MaximumPosition) are calculated relative to the position zero and not to the current position. So before executing a PT trajectory, the user must pay attention to the current position of the positioners to make sure that the trajectory will not exceed the positioner travel limits.
- If all is OK, it returns “SUCCESS” (0). Otherwise, it returns a corresponding error.

---

#### NOTES

**Because of the PT trajectory internal calculation of elements end velocity, a correct PT trajectory file must have at least two lines with zero displacements at the trajectory end. Otherwise, the “MultipleAxesPTVerification” function returns error (-70).**

**The “MultipleAxesPTVerification” function is independent from the “MultipleAxesPTExecution” function, but it is highly recommended to execute this function before executing the PT trajectory.**

---

### **Prototype**

```
int MultipleAxesPTVerification(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250]  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -61: Error file corrupt or file doesn't exist.
- -66: Trajectory doesn't content any element.
- -68: Acceleration on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -70: Final velocity on trajectory is not zero.
- -72: Error trajectory initialization.
- -75: Trajectory element has a negative or null delta T.

### **6.2.1.122 `MultipleAxesPTVerificationResultGet`**

#### **Name**

**MultipleAxesPTVerificationResultGet** – Gets the results of the “MultipleAxesPTVerification” function.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name length (must  $\leq 250$ ): (-3)
- Check positioner name: (-18)
- Check the last MultipleAxes PTVerification (must be done): (-22)

#### **Description**

This function returns the results of the previous “MultipleAxesPTVerification” function, for every positioner. The results are the travel requirements (min and max values), the possible maximum velocity and the possible maximum acceleration.

If no verification was previously done , error (-22) is returned.

#### **Prototype**

```
int MultipleAxesPTVerificationResultGet(  
    int SocketID,  
    char PositionerName[250],  
    char * TrajectoryFileName,  
    double * MinimumPosition,  
    double * MaximumPosition,  
    double * MaximumVelocity,  
    double * MaximumAcceleration  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

TrajectoryFileName	char *	Examined trajectory file name.
MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
MaximumVelocity	double *	Maximum velocity (units/s).
MaximumAcceleration	double *	Maximum acceleration (units/s <sup>2</sup> ).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.123 `MultipleAxesPVTExecution`**

#### **Name**

**MultipleAxesPVTExecution** – Executes a PVT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length: (-3)
- Check group type (must be a MultipleAxes group): (-8)
- Check input value (number of executions must >0): (-17)
- Check group name: (-19)
- Group state must be “READY”: (-22)
- Check backlash (must not be enabled): (-46)
- Check BaseVelocity (stages.ini, must = 0): (-48)
- Check trajectory file existence or file reading: (-61)
- Check message queue: (-71)

#### **Description**

This function executes a PVT (Position Velocity Time) trajectory. The trajectory file must be stored in “Admin\Public\Trajectory” folder of XPS controller. If the trajectory cannot be initialized (message queue or task error), error (-72) is returned.

Before the trajectory execution, it is recommended to check whether the trajectory is within the positioner motion capabilities by using “MultipleAxesPVTVerification” and “MultipleAxesPVTVerificationResultGet” functions.

During the trajectory execution, if a positioner reaches one of its travel limits, the trajectory execution stops and error (-25) error is generated under positioner errors.

#### **NOTE**

**In case of errors (-33), (-25) and (-44), , the group state changes to DISABLE. To help determine the error source, check the positioner errors, the hardware status and the driver status.**

#### **Prototype**

```
int MultipleAxesPVTExecution
{
    int SocketID,
    char GroupName[250],
    char FileName[250],
    int ExecutionNumber
}
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
ExecutionNumber	int	Number of trajectory executions.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -46: Not allowed action due to backlash.
- -48: BaseVelocity must be null.
- -61: Error file corrupt or file doesn't exist.
- -71: Error read from or write in message queue.
- -72: Error trajectory initialization.

### **6.2.1.124 `MultipleAxesPVTLoadToMemory`**

#### **Name**

**MultipleAxesPVTLoadToMemory** – Loads some lines of PVT trajectory to the controller memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function loads some lines of PVT trajectory into XPS controller memory. Each trajectory element must be separated by a comma. The trajectory lines are separated between each other by “\n” (LF) character. To verify or to execute the PVT trajectory loaded in memory, use “**FromMemory**” string instead of a file name.

#### **NOTES**

All of the PVT functions, when called with the string “FromMemory” instead of a FileName, will perform the same operation as the PVT trajectory in RAM as it does from a disk.

##### *Example:*

```
MultipleAxesPVTLoadToMemory(socketId,myGroup,"dT1,dX11,Vout11,dX12,Vout
12\n...dTn,dXn1,Voutn1,dXn2,Voutn2\n")
MultipleAxesPVTVerification (socketId,myGroup,FromMemory)
MultipleAxesPVTExecution(socketId,myGroup,FromMemory,I).
```

#### **Prototype**

```
int MultipleAxesPVTLoadToMemory(
    int SocketID,
    char GroupName[250],
    char TrajectoryData[400]
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
TrajectoryData	char *	Trajectory data lines.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### **6.2.1.125 `MultipleAxesPVTParametersGet`**

#### **Name**

`MultipleAxesPVTParametersGet` – Gets PVT trajectory parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Check current executing trajectory type (must be PVT): (-22)

#### **Description**

This function returns the PVT trajectory parameters (trajectory name and current executing element number) of the current PVT trajectory.

#### **Prototype**

```
int MultipleAxesPVTParametersGet(  
    int SocketID,  
    char GroupName[250],  
    char * FileName,  
    int * CurrentElementNumber  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

FileName	char *	Currently executing trajectory file name.
CurrentElementNumber	int *	Currently executing element number.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.126 `MultipleAxesPVT_PulseOutputGet`**

#### **Name**

**MultipleAxesPVT\_PulseOutputGet** – Gets the configuration of pulse generation for PVT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)

#### **Description**

This function returns the last configuration of pulse generation for a PVT trajectory, that was previously set by *MultipleAxesPVT\_PulseOutputSet()*.

The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

#### ***Example:***

*MultipleAxesPVT\_PulseOutputSet(MyGroup, 3, 5, 0.01)*

*MultipleAxesPVT\_PulseOutputGet(MyGroup) => 0,3,5,0.01 (0 is error return, means OK)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

Start element= 3

End element = 5

Time interval = 0.01 seconds.

#### **Prototype**

```
int MultipleAxesPVT_PulseOutputGet(
    int SocketID,
    char GroupName[250],
    int * StartElement,
    int * EndElement,
    double * TimeInterval
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

StartElement	int *	Start pulse element number.
EndElement	int *	End pulse element number.
TimeInterval	double *	Time interval between pulses (seconds).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.

### **6.2.1.127 `MultipleAxesPVT_PulseOutputSet`**

#### **Name**

**MultipleAxesPVT\_PulseOutputSet** – Sets the configuration of pulse generation for PVT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Check the pulse generation must not be in progress: (-22)

#### **Description**

This function configures and activates the pulse generation of a PVT trajectory. The pulse generation is defined by a start element, an end element, and a time interval in seconds. If a pulse generation is already activated on the selected PVT trajectory , error (-22) is returned

Please note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, it is required to define the pulse output settings again.

This capability allows output of pulses at constant time intervals on a PVT trajectory. The pulses are generated between the first and the last trajectory elements. The minimum possible time interval is CorrectorISRPeriod value (*system.ref*).

The trajectory pulses are generated on the following GPIO outputs:

GPIO signals	ISA XPS controller	PCI XPS controller (for example XPS-RL) with basic GPIO board	PCI XPS controller (for example XPS-RL) with extended GPIO board
Window	GPIO2, pin 11	GPIO1.DO6	GPIO5.DO14
Pulses	GPIO2, pin 12	GPIO1.DO7	GPIO5.DO15

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual

#### ***Example:***

*MultipleAxesPVT\_PulseOutputSet(Group1, 3, 5, 0.01)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

#### **Prototype**

```
int MultipleAxesPVT_PulseOutputSet(
    int SocketID,
    char GroupName[250],
    int StartElement,
    int EndElement,
    double TimeInterval
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
StartElement	int	Start pulse element number.
EndElement	int	End pulse element number.
TimeInterval	double	Time interval between pulses (seconds).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.128 MultipleAxesPVTResetInMemory

#### Name

**MultipleAxesPVTResetInMemory** – Deletes the content of the PVT trajectory buffer in the controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function deletes the PVT trajectory buffer, that was previously loaded with the “MultipleAxesPVTLoadToMemory” function, from the controller memory.

#### Prototype

```
int MultipleAxesPVTResetInMemory(  
    int SocketID,  
    char GroupName[250]  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### **6.2.1.129 `MultipleAxesPVTVerification`**

#### **Name**

**MultipleAxesPVTVerification** – Checks the PVT trajectory data file.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length (must  $\leq 250$ ): (-3)
- Check group type (must be a MultipleAxes group): (-8)
- Check group name: (-19)
- Check BaseVelocity value (must = 0): (-48)
- Check trajectory file existence and the file format: (-61)
- Check trajectory (number of elements must  $> 0$ ): (-66)
- Check velocity (Minimum Velocity  $\leq$  Velocity  $\leq$  Maximum Velocity): (-68)
- Check acceleration (Minimum acc.  $\leq$  acceleration  $\leq$  Maximum acc.): (-69)
- Check end output velocity (must = 0): (-70)
- Check delta time (DeltaTime must  $> 0$ ): (-75)

#### **Description**

This function verifies the execution of the PVT trajectory. The results of the verification can be obtained by “`MultipleAxesPVTVerificationResultGet`” function. The trajectory file must be stored in “`\ADMIN\Public\Trajectory`” folder of the XPS controller. If the trajectory cannot be initialized (task error), error (-72) is returned.

This function can be executed at any time and is independent of the trajectory execution. It performs the following:

- Checks the trajectory file for data coherence.
- Calculates the trajectory limits, 1) the required travel per positioner, 2) the maximum possible trajectory velocity and 3) the maximum possible trajectory acceleration. This function helps to define the parameters for the trajectory execution.
- The required travel values (MinimumPosition and MaximumPosition) are calculated relative to the position zero and not to the current position. So before executing a PVT trajectory, the user must pay attention to the current position of the positioners to make sure that the trajectory will not exceed the positioner travel limits.
- If all is OK, it returns “`SUCCESS`” (0). Otherwise, it returns a corresponding error.

---

#### **NOTE**

The “`MultipleAxesPVTVerification`” function is independent from the “`MultipleAxesPVTExecution`” function, but it is highly recommended to execute this function before executing a PVT trajectory.

---

### **Prototype**

```
int MultipleAxesPVTVerification(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250]  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -61: Error file corrupt or file doesn't exist.
- -66: Trajectory doesn't content any element.
- -68: Acceleration on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -70: Final velocity on trajectory is not zero.
- -72: Error trajectory initialization.
- -75: Trajectory element has a negative or null delta T.

### **6.2.1.130 `MultipleAxesPVTVerificationResultGet`**

#### **Name**

**MultipleAxesPVTVerificationResultGet** – Gets the results of the “MultipleAxesPVTVerification” function.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name length (must  $\leq 250$ ): (-3)
- Check positioner name: (-18)
- Check the last MultipleAxes PVTVerification (must be done): (-22)

#### **Description**

This function returns the results of the previous “MultipleAxesPVTVerification” function, for every positioner. The results are the travel requirements (min and max values), the possible maximum velocity and the possible maximum acceleration.

If no verification was previously done , error e (-22) is returned.

#### **Prototype**

```
int MultipleAxesPVTVerificationResultGet(
```

```
    int SocketID,
    char PositionerName[250],
    char * TrajectoryFileName,
    double * MinimumPosition,
    double * MaximumPosition,
    double * MaximumVelocity,
    double * MaximumAcceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

TrajectoryFileName	char *	Examined trajectory file name.
MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
MaximumVelocity	double *	Maximum velocity (units/s).
MaximumAcceleration	double *	Maximum acceleration (units/s <sup>2</sup> ).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.131 OpenConnection

#### Name

**OpenConnection** – opens a socket to connect to TCP server (local).

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check number of used sockets (Max = 100): if no free socket then the SocketID is set to -1.

#### Description

This function opens a socket in a TCL script located in the “Scripts” directory of the XPS controller.

The TCP/IP communication is configured as:

Local Host Address = 127.0.0.1

IP Port = 5001

This function returns a socket identifier to use for each function call. The socket identifier is defined between 0 to 99. If the TCP/IP connection fails then the “SocketID” value is -1.

#### Prototype

```
int OpenConnection(  
    int TimeOut,  
    int SocketID  
)
```

#### Input parameters

TimeOut	int	Timeout in seconds used for each function execution.
SocketID	int	Socket identifier used in each function.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.132 PositionerAccelerationAutoScaling

#### Name

**PositionerAccelerationAutoScaling** –Executes Auto-scaling process to determine the stage scaling acceleration.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner”: (-8)
- Check group type: (-8)
- Check positioner name: (-18)
- Group status must be not initialized: (-22)
- Control loop type must be “PIDFFAcceleration”: (-24)

#### Description

This function executes an auto-scaling process and returns the calculated scaling acceleration. The selected group must be in “NOTINIT” state, otherwise error (-22) is returned.

It only works, if the positioner control loop type is “PIDFFAcceleration” (acceleration control), otherwise error (-24) is returned

This function checks the positioner error. If an error is detected, the hardware status register is reset (motor on) and the positioner error is cleared before checking it again. If a positioner error is present, the motor is turned off, error (-5) is returned and the group status becomes “NOTINIT”.

If there is no positioner error, then the master-slave error is cleared, the encoder is preset (update encoder position) and the user travel limits are checked. If a travel limit error is detected then the motor is turned off, error (-35) is returned and the group status becomes “NOTINIT”.

If no error is detected , the motor initializes. If motor initialization fails, the error (-50) is returned and the group status becomes “NOTINIT”.

If motor initialization is successful, the positions are preset, the motion is enabled (the motor is powered) permitting the process to auto-scale if the motion cannot be enabled, error (-22) is returned.

If the auto-scaling fails error (-105) is returned or if the motion becomes disabled, error (-26) is returned.

The auto-scaling process is executed in 5 periods. At the end of each period, the auto-scaling process estimates the auto- scaling quality by calculating the signal to noise ratio. If it is very close to zero, it means there is no oscillation, so error (-101) is returned. if the signal to noise ratio > MaximumNoiseRatio defined in system.ref (normally between 0.1 and 0.2), error (-102) is returned.

If the number of acquired data points (minimum = 9) or the number of acquired signal periods (minimum = 5) is not enough for a good estimate then error (-103) is returned.

At the end of this function, the new value of scaling acceleration is returned and the group status becomes “NOTINIT” once again.

### **Prototype**

```
int PositionerAccelerationAutoScaling(  
    int SocketID,  
    char * PositionerName,  
    double * Scaling  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Name of a positioner.
FrequencyTicks	int	Number of frequency ticks.

### **Output parameters**

Scaling	double *	Calculated scaling acceleration value.
---------	----------	--

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -5: Not allowed due to a positioner error or hardware status.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).
- -35: Position is outside of travel limits.
- -50: Motor initialization error. Check InitializationAccelerationLevel, ScalingAcceleration, MaximumJerkTime, EncoderResolution or EncoderScalePitch.
- -101: Relay Feedback Test failed: No oscillation.
- -102: Relay Feedback Test failed: Signal too noisy.
- -103: Relay Feedback Test failed: Signal data not enough for analyse.
- -105: Error of scaling calibration initialization.

### **6.2.1.133 PositionerAnalogTrackingPositionParametersGet**

#### **Name**

**PositionerAnalogTrackingPositionParametersGet** – Gets the parameters of the current tracking position mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter: (-8), (-18)

#### **Description**

This function returns the current analog input name, the current offset and the current scale used by analog tracking position mode.

#### **NOTE**

**“velocity” and “acceleration” define the maximum velocity and acceleration used in the position tracking mode.**

#### **Prototype**

```
int PositionerAnalogTrackingPositionParametersGet(
    int SocketID,
    char * FullPositionerName,
    char * GPIOName,
    double * Offset,
    double * Scale,
    double * Velocity,
    double * Acceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name FrequencyTicks.

#### **Output parameters**

GPIOName	char	Analog input name (ADC).
Offset	double *	Offset in volts.
Scale	double *	Scale (Units/Volts).
Velocity	double *	Velocity (Units/s).
Acceleration	double *	Acceleration (Units/s <sup>2</sup> ).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.134 PositionerAnalogTrackingPositionParametersSet**

#### **Name**

**PositionerAnalogTrackingPositionParametersSet** – Sets the parameters of the current tracking position mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Positioner and GPIO type (ADC): (-8)
- Check velocity and acceleration: (-17)
- Check input parameter: (-18)

#### **Description**

This function modifies the analog input name, the offset and the scale used by the analog tracking position mode. To use this function, the group state must be READY otherwise error (-22) is returned.

The “Offset” and the “Scale” parameters are used to calculate the target tracking position:

$$\text{TrackingPosition} = \text{InitialPosition} + (\text{AnalogValue} - \text{Offset}) * \text{Scale}$$

The “velocity” and “acceleration” parameters define the maximum velocity and acceleration used in the position tracking mode.

#### **NOTE**

**The parameters for analog tracking position mode can be reset, if the “GPIOName” parameter is blank.**

#### **Prototype**

```
int PositionerAnalogTrackingPositionParametersSet(
    int SocketID,
    char FullPositionerName,
    char * GPIOName,
    double Offset,
    double Scale,
    double Velocity,
    double Acceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char	Positioner name FrequencyTicks.
GPIOName	char *	Analog input name (ADC).
Offset	double	Offset in volts.
Scale	double	Scale (Units/Volts).
Velocity	double	Velocity (Units/s).
Acceleration	double	Acceleration (Units/s <sup>2</sup> ).

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.135 PositionerAnalogTrackingVelocityParametersGet**

#### **Name**

**PositionerAnalogTrackingVelocityParametersGet** – Gets the parameters of the current tracking velocity mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- ”.
- Check input parameter: (-8), (-18)

#### **Description**

This function returns the analog input name, the offset, the scale, the deadband threshold and the order used by analog traking velocity mode.

#### **NOTE**

**“velocity” and “acceleration” define the maximum velocity and acceleration used in the velocity tracking mode.**

#### **Prototype**

```
int PositionerAnalogTrackingVelocityParametersGet(
    int SocketID,
    char FullPositionerName,
    char * GPIOName,
    double * Offset,
    double * Scale,
    double * DeadBandThreshold,
    int * Order,
    double * Velocity,
    double * Acceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name FreuecyTicks.

#### **Output parameters**

GPIOName	char *	Analog input name (ADC).
Offset	double *	Offset in volts.
Scale	double *	Scale (Units/Volts).
DeadBandThreshold	double *	Dead band threshold (Volts).
Order	integer *	Order (No unit).
Velocity	double *	Velocity (Units/s).
Acceleration	double *	Acceleration (Units/s <sup>2</sup> ).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.136 PositionerAnalogTrackingVelocityParametersSet**

#### **Name**

**PositionerAnalogTrackingVelocityParametersSet** – Sets the parameters of tcurrent tracking velocity mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check GPIO type (ADC): (-8)
- Check Positioner: (-8), (-18)
- Check velocity and acceleration: (-17)

#### **Description**

This function allows modifying the GPIO name, offset, scale, deadband threshold and the order used by analog tracking velocity mode. To use this function the group state must be READY, otherwise error (-22) is returned.

The “velocity” and “acceleration” define the maximum velocity and acceleration used in the velocity tracking mode.

The target tracking velocity is defined as follows:

$$\text{InputValue} = \text{GPIOAnalogInput} - \text{Offset}$$

$$\text{MaxADCAmplitude} = 10/\text{GPIOAnalogGain}$$

if (InputValue  $\geq 0$ ) then

$$\text{InputValue} = \text{InputValue} - \text{DeadBandThreshold}$$

if (InputValue < 0) then InputValue = 0

else

$$\text{InputValue} = \text{AnalogInputValue} + \text{DeadBandThreshold}$$

if (InputValue > 0) then InputValue = 0

$$\text{OutputValue} = (|\text{InputValue}|/\text{MaxADCAmplitude})^{\text{Order}}$$

$$\text{TrackingVelocity} = \text{Sign}(\text{InputValue}) * \text{OutputValue} * \text{Scale} * \text{MaxADCAmplitude}$$

#### **NOTE**

The analog tracking velocity mode can be reset if the “GPIOName” parameter is blank.

#### **Prototype**

```
int PositionerAnalogTrackingVelocityParametersSet(
    int SocketID,
    char * FullPositionerName,
    char * GPIOName,
    double Offset,
    double Scale,
    double DeadBandThreshold,
    int Order,
    double Velocity,
    double Acceleration
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
FullPositionerName	char *	Positioner name FrequencyTicks.
GPIOName	char *	Analog input name (ADC).
Offset	double	Offset in volts.
Scale	double	Scale (Units/Volts).
	double	Dead band threshold (Volts).
Order	integer	Order (No unit).
Velocity	double	Velocity (Units/s).
Acceleration	double	Acceleration (Units/s <sup>2</sup> ).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.137 PositionerBacklashDisable

#### Name

**PositionerBacklashDisable** – Disables the backlash compensation.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)

#### Description

This function disables the backlash compensation.

In the “stages.ini” file the parameter “Backlash” will enable or disable this feature as follows:

Backlash = 0 —>Disable backlash  
Backlash >0 —>Enable backlash

---

#### NOTE

**The backlash compensation is not allowed with a secondary positioner (gantry mode).**

**The backlash must be disabled to execute a trajectory, use a jog mode or analog tracking mode.**

---

#### Prototype

```
int PositionerBacklashDisable(  
    int SocketID,  
    char * FullPositionerName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.138 PositionerBacklashEnable**

#### **Name**

**PositionerBacklashEnable** – Enables the backlash compensation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Group status must be “NOTINIT”: (-22)
- Check the positioner type (must not be a secondary positioner): (-8)

#### **Description**

This function enables the backlash compensation defined in the “stages.ini” file or by “PositionerBacklashSet” function. If the backlash compensation value is null then this function will have no effect, and backlash compensation will remain disabled..

The group state must be NOTINIT to enable the backlash compensation. If it is not the case error (-22) is returned.

The parameter “Backlash in the “stages.ini” file” allows the user to enable or disable the backlash compensation.

Backlash = 0 —>Disable backlash

Backlash >0 —>Enable backlash

#### **NOTE**

**The backlash must be disabled to execute a trajectory use the jog mode or analog tracking mode.**

#### **CAUTION**



**It is not possible to use backlash compensation with positioners that have one of the following:**

- 1)**“HomeSearchSequenceType” defined as “CurrentPositionAsHome”**
- 2)**“PositionerMappingFileName” defined in the stages.ini file.**

#### **Prototype**

```
int PositionerBacklashEnable(
    int SocketID,
    char * FullPositionerName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -22: Not allowed action.

### 6.2.1.139 PositionerBacklashGet

#### Name

**PositionerBacklashGet** – Gets the backlash compensation value.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)

#### Description

This function returns the backlash compensation value, defined in the “stages.ini” file or by “PositionerBacklashSet” function, and the backlash status (“Enable” or “Disable”).

#### Prototype

```
int PositionerBacklashGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * BacklashValue,  
    char * Status  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

BacklashValue	double *	Backlash compensation value (units).
Status	char *	Backlash status (“Enable” or “Disable”).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.140 PositionerBacklashSet**

#### **Name**

**PositionerBacklashSet** – Sets the backlash compensation value.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)
- The “BacklashValue” must be positive: (-17)

#### **Description**

This function changes the backlash compensation value.

#### **NOTE**

This function can be used only if a backlash compensation is defined in “stages.ini” file (Backlash >0) , otherwise error (-22) is returned.

#### **Prototype**

```
int PositionerBacklashSet(
    int SocketID,
    char * FullPositionerName,
    double BacklashValue
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
BacklashValue	double	Backlash compensation value (units).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.

### **6.2.1.141 PositionerCompensatedFastPCOAbort**

#### **Name**

**PositionerCompensatedFastPCOAbort** – Aborts the CIE fast compensated PCO pulses generation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)

#### **Description**

This function aborts the CIE fast compensated PCO pulses generation. The pulses generation is stopped immediately; no more pulses will be generated even if the scanning positioner continues to move across the predefined firing positions. To stop the scanning move, use GroupMoveAbort() function.

#### **Prototype**

```
int PositionerCompensatedFastPCOAbort(  
    int SocketID,  
    char * FullPositionerName  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.142 PositionerCompensatedFastPCOCurrentStatusGet**

#### **Name**

**PositionerCompensatedFastPCOCurrentStatusGet** – Gets current status of CIEFAST compensated PCO pulses generation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check if CIEFAST compensated PCO pulses generation is enabled: (-121)

#### **Description**

This function gets the current status of CIEFAST compensated PCO pulses generation.

Status possible values:

- 0: Pulses generation inactive (idle, no error)  
1: Pulses generation activated (running)  
-1: Pulses generation aborted with errors.

#### **Prototype**

```
int PositionerCompensatedFastPCOCurrentStatusGet(  
    int SocketID,  
    char * FullPositionerName,  
    int * Status  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

Status	int *	PCO pulses generation status.
--------	-------	-------------------------------

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Function is not allowed due to configuration disabled.

### **6.2.1.143 PositionerCompensatedFastPCOEnable**

#### **Name**

**PositionerCompensatedFastPCOEnable** – Activates the CIEFAST compensated PCO pulses generation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check if CIEFAST compensated PCO pulses generation is enabled: (-121)

#### **Description**

This function activates the CIE Fast compensated PCO pulses generation (status becomes running (value 1)). The pulses will be generated when the scanning positioner moves across the predefined positions. When the last pulse is generated, the CIE Fast compensated PCO mode will become inactive (status becomes inactive (value 0)). To get status of the CIE Fast compensated PCO pulses generation, use **PositionerCompensatedFastPCOCurrentStatusGet()** function.

Note that only the scanning positioner positions are used to fire pulses: if you prepare a set of positions at a given location and then enable the pulses generation and start the move from a different location, the pulses could be generated but their accuracy will be impacted by the mapping difference between the two locations.

This function must be used after the firing pulses data preparation with the **PositionerCompensatedFastPCOPrepare()**, elsewhere the function fails and the error (-122) will be returned.

#### **Prototype**

```
int PositionerCompensatedFastPCOEnable(
    int SocketID,
    char * FullPositionerName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -115: Function is not supported by current hardware.
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### **6.2.1.144 PositionerCompensatedFastPCOFromFile**

#### **Name**

**PositionerCompensatedFastPCOFromFile** – Reads firing positions from a data file to controller's memory.

#### **Input tests**

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the positioner name: (-18)
- Check if CIEFAST compensated PCO pulses generation is enabled: (-121)

#### **Description**

This function reads firing positions from a data file to the controller's memory.

The data file contains lines of data, formatted as follows:

Position<sub>i</sub>

#### *Example:*

Position<sub>1</sub>

Position<sub>2</sub>

... ...

Position<sub>N</sub>

Data conditions:

Position<sub>i</sub> > Position<sub>i-1</sub>, Width<sub>i</sub> < Position<sub>i+1</sub> - Position<sub>i</sub>.

#### **NOTE**

Position<sub>i</sub> (i = 1...N) are the offset values relative to the scanning positioner start position that is defined in the **PositionerCompensatedFastPCOPrepare()**.

#### **Prototype**

```
int PositionerCompensatedFastPCOFromFile(
    int SocketID,
    char * PositionerName,
    char * DataFileName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
PositionerName	char *	Positioner name.
DataFileName	char *	Data file name.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -61: Error file corrupt or file doesn't exist.
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### **6.2.1.145 PositionerCompensatedFastPCOLoadToMemory**

#### **Name**

**PositionerCompensatedFastPCOLoadToMemory**– Appends firing positions to controller memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check if CIEFAST compensated PCO pulses generation is enabled: (-121)

#### **Description**

This function appends firing positions to controller memory from input parameters.

To reset the controller memory, the PositionerCompensatedFastPCOMemoryReset() function is provided.

The data line format can be a single or several offset positions as in PositionerCompensatedFastPCOSet() function:

#### ***Example:***

```
Positioni
Positioni Positioni+1 Positioni+2
PositionerCompensatedLoadToMemory(XY.X, 0 0.1 0.2 0.3 0.4 0.5)
```

#### **Prototype**

```
int PositionerCompensatedFastPCOLoadToMemory(
    int SocketID,
    char * PositionerName,
    char * DataLine
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
DataLine	char *	Some data lines.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### 6.2.1.146 PositionerCompensatedFastPCOMemoryReset

#### Name

**PositionerCompensatedFastPCOMemoryReset** – Resets CIEFast compensated PCO data memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check if CIEFAST compensated PCO pulses generation is enabled: (-121)

#### Description

This function resets the CIE fast compensated PCO data memory. This function is useful to remove the data that was previously entered with the PositionerCompensatedFastPCOLoadToMemory() function.

#### Prototype

```
int PositionerCompensatedFastPCOMemoryReset(  
    int SocketID,  
    char * PositionerName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Function is not allowed due to configuration disabled.

### **6.2.1.147 PositionerCompensatedFastPCOPrepare**

#### **Name**

**PositionerCompensatedFastPCOPrepare** – Prepares data for CIEFast compensated PCO pulses generation.

#### **Input tests**

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check direction value: (-17)
- Check the positioner name: (-18)
- Check if first and last PCO positions are within positions limits: (-35)
- Check if CIEFAST compensated PCO pulses generation is enabled: (-121)

#### **Description**

This function calculates the firing for absolute positions, *in user's coordinate system* and converts them to firing absolute raw PCO positions, *in encoder's coordinate system*. It will use the supplied start positions and the offset positions set with *PositionerCompensatedFastPCOSet()* or *PositionerCompensatedFastPCOFromFile()* or *PositionerCompensatedFastPCOLoadToMemory()* function.

When mappings are enabled, the correction between user's coordinate system position and raw encoder position will be different at each location. For this reason, the prepare function must know the location (*positions of all positioners in the scanning group*) where the scan will be done.

This function must be called before the use of *PositionerCompensatedFastPCOEnable()* function.

Parameters:

- ScanDirection: Scan direction,(value: 1 (*positive*) or -1 (*negative*)).
- StartPosition1: Group 1<sup>st</sup> positioner start position
- StartPosition2: Group 2<sup>nd</sup> positioner start position
- StartPosition3: Group 3<sup>rd</sup> positioner start position
- ..... etc .....

#### **Prototype**

```
int PositionerCompensatedFastPCOPrepare(  
    int SocketID,  
    char * PositionerName  
    int ScanDirection,  
    double StartPosition1,  
    double StartPosition2,  
    double StartPosition3,  
    ...  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
ScanDirection	int	Scan direction (1 or -1).
StartPosition1	double	Group 1 <sup>st</sup> positioner start position (units).
StartPosition2	double	Group 2 <sup>nd</sup> positioner start position (units).
StartPosition3	double	Group 3 <sup>rd</sup> positioner start position (units).
....		

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -35: Position is outside of travel limits.
- -115: Function is not supported by current hardware.
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### **6.2.1.148 PositionerCompensatedFastPCOPulseParametersGet**

#### **Name**

**PositionerCompensatedFastPCOPulseParametersGet** – Gets pulse configuration to compensated PCO.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check pulses generation status: (-22)
- Check if CIEFAST compensated PCO pulses generation is supported: (-115)

#### **Description**

This function returns the set values of parameters: Pulse width, Pulse polarity and Pulse toggle.

#### **Prototype**

```
int PositionerCompensatedFastPCOPulseParametersGet(
    int SocketID,
    char * PositionerName,
    double * PulseWidth,
    int * PulsePolarity,
    bool * PulseToggle
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

PulseWidth	double *	Width of pulse enable signal (units).
PulsePolarity	int *.	
PulseToggle	bool *.	

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -115: Function is not supported by current hardware.

### **6.2.1.149 PositionerCompensatedFastPCOPulseParametersSet**

#### **Name**

**PositionerCompensatedFastPCOPulseParametersGet** – Sets pulse configuration to compensated PCO.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check pulses generation status: (-22)
- Check if CIEFAST compensated PCO pulses generation is supported: (-115)

#### **Description**

This function sets values of parameters: Pulse width, Pulse polarity and Pulse toggle.

#### **Prototype**

```
int PositionerCompensatedFastPCOPulseParametersGet(
```

```
    int SocketID,
    char * PositionerName,
    double PulseWidth,
    int PulsePolarity,
    bool PulseToggle
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PulseWidth	double *	Width of pulse enable signal (units).
PulsePolarity	int *.	
PulseToggle	bool *.	

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -115: Function is not supported by current hardware.

### **6.2.1.150 PositionerCompensatedFastPCOSet**

#### **Name**

**PositionerCompensatedFastPCOSet** – Calculates a set of evenly spaced firing positions to the controller memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Start <Stop, Step >0, and (Stop – Start) <Step: (-17)
- Check the positioner name: (-18)
- Check if CIEFAST compensated PCO pulses generation is enabled: (-121)
- Check data number NData = floor(Stop - Start) / Step + 1, if NData >1000000: (-122)

#### **Description**

This function calculates a set of evenly spaced firing positions to the controller memory.

Parameters:

- Start: Relative offset where first PCO trigger is generated. Start must be less than Stop.
- Stop: Relative offset where last PCO pulses can be generated.
- Step: Distance between two consecutive pulses. Step must be positive.

This function will create a table of points that has as first point Start value and last point Stop value:

Index = 0	Start
Index = 1	Start + Step
Index = 2	Start + 2 * Step
Index = 3	Start + 3 * Step
Index = 4	Start + 4 * Step
Index = ...	...
Index = N	Start + N * Step

#### ***Example:***

Send PositionerCompensatedFastPCOSet(XY.X, 0, 10, 1) will generate points table below:

Index = 0	0
Index = 1	1
Index = 2	2
Index = 3	3
Index = 4	4
Index = ...	...
Index = 10	10

### **Prototype**

```
int PositionerCompensatedFastPCOSet(  
    int SocketID,  
    char * PositionerName,  
    double Start,  
    double Stop,  
    double Step  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
Start	double	Start position (units).
Stop	double	Stop position (units).
Step	double	Distance between two consecutive pulses (units).

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### **6.2.1.151 PositionerCompensatedPCOAbort**

#### **Name**

**PositionerCompensatedPCOAbort** – Aborts the CIE08 compensated PCO pulses generation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check the positioner name: (-18)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the CIE board supports this function: (-115)
- Check CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)

#### **Description**

This function aborts the CIE08 compensated PCO pulses generation. The pulses generation is stopped immediately; no more pulses will be generated even if the scanning positioner continues to move across the predefined firing positions. To stop the scanning move, use *GroupMoveAbort()* function.

#### **NOTE**

The function works only when the CIE08 compensated PCO mode configuration is enabled (*system.ini*: CIE08CompensatedPCOMode = Enabled).

This function can be used only with a position encoder (“AquadB” or “AnalogInterpolated”), elsewhere ERR\_UNCOMPATIBLE (-24) error is returned.

#### **Prototype**

```
int PositionerCompensatedPCOAbort(
    int SocketID,
    char * FullPositionerName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner name doesn't exist or unknown command.
- -24: Incorrect file name, bad section name, or not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware
- -121: Function is not allowed due to configuration disabled.

### **6.2.1.152 PositionerCompensatedPCOCurrentStatusGet**

#### **Name**

**PositionerCompensatedPCOCurrentStatusGet** – Gets current status of CIE08 compensated PCO pulses generation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check the positioner type: (-8)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the CIE board supports this function: (-115)
- Check CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)

#### **Description**

This function gets the current status of CIE08 compensated PCO pulses generation.

Status possible values:

- 0: Pulses generation inactive (idle, no error)
- 1: Pulses generation activated (running)
- -1: Pulses generation aborted with errors.

#### **Prototype**

```
int PositionerCompensatedPCOCurrentStatusGet(
    int SocketID,
    char * FullPositionerName,
    int * Status
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

Status	int *	PCO pulses generation status.
--------	-------	-------------------------------

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner name doesn't exist or unknown command.
- -24: Incorrect file name, bad section name, or not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware
- -121: Function is not allowed due to configuration disabled.

### **6.2.1.153 PositionerCompensatedPCOEnable**

#### **Name**

**PositionerCompensatedPCOEnable** – Activates the CIE08 compensated PCO pulses generation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the CIE board supports this function: (-115)
- Check CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)
- Check if data have been prepared by PositionerCompensatedPCOPrepare(): (-122)
- Check current position is out and at the good side of scanning zone (left of scanning zone if ScanDirection positive, right of scanning zone if ScanDirection negative): (-22)
- Check CIE08 compensated PCO mode is running: (-22)

#### **Description**

This function activates the CIE08 compensated PCO pulses generation (*status becomes running (value 1)*). The pulses will be generated when the scanning positioner moves across the predefined positions. When the last pulse is generated, the CIE08 compensated PCO mode will become inactive (*status becomes inactive (value 0)*). To get status of the CIE08 compensated PCO pulses generation, use *PositionerCompensatedPCOCurrentStatusGet()* function.

Note that only the scanning positioner positions are used to fire pulses: if you prepare a set of positions at a given location and then enable the pulses generation and start the move from a different location, the pulses could be generated but their accuracy will be impacted by the mapping difference between the two locations.

This function must be used after the firing pulses data preparation with the *PositionerCompensatedPCOPrepare()*, elsewhere the function fails and the error **ERR\_CHECK\_DATA\_INCORRECT** (-122) will be returned. With XPS-Q if the settings are out of range, the user might experience pulses spaced unevenly without getting any error messages.

---

#### **NOTE**

**The PCO pulses generation depends on the ScanVelocity and the pulse settling time (set by *PositionerPositionComparePulseParametersSet()*)**

Valid settings are shown in the table below:

<b>Pulse settling time (μs)</b>	<b>PCO encoder frequency (kHz)</b>			
	25	50	125	> 500
0.075			OK	OK
1		OK	OK	
4	OK	OK		
12	OK			

How to determine the PCO encoder frequency :

- For AquadB encoder :  
PCO encoder frequency = Velocity / EncoderResolution
- For analog interpolated encoder :  
PCO encoder frequency = Velocity \* HardInterpolatorFactor / EncoderScalePitch

**Example:** XML310 stage (EncoderScalePitch=0.004 mm, HardInterpolatorFactor=200). With ScanVelocity=10mm/s => PCO encoder frequency = 10\*200/0.004 = 500 kHz

#### NOTE

The function works only when the CIE08 compensated PCO mode configuration is enabled (*system.ini*: CIE08CompensatedPCOMode = Enabled).

This function can be used only with a position encoder ("AquadB" or "AnalogInterpolated"), otherwise ERR\_UNCOMPATIBLE (-24) error is returned.

#### Prototype

```
int PositionerCompensatedPCOEnable(
    int SocketID,
    char * FullPositionerName
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
FullPositionerName	char *	Positioner name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Incorrect file name, bad section name, or not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware.
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### 6.2.1.154 PositionerCompensatedPCOFromFile

#### Name

**PositionerCompensatedPCOFromFile** – Reads firing positions from a data file to controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check the positioner type: (-8)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the CIE board supports this function: (-115)
- Check CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)
- Check data file exists: (-61)
- Check data from file (must be  $\text{Position}_i > \text{Position}_{i-1}$ ,  $\text{Width}_i < \text{Position}_{i+1} - \text{Position}_i$ ): (-122)

#### Description

This function reads firing positions from a data file to the controller memory.

The data file contains lines of data, formatted as follows:

$\text{Position}_i < \text{Space or Tabulation} > \text{Width}_i < \text{CRLF or LF} >$

#### *Example :*

```
Position1Width1
Position2Width2
...
PositionNWidthN
```

Data conditions:  $\text{Position}_i > \text{Position}_{i-1}$ ,  $\text{Width}_i < \text{Position}_{i+1} - \text{Position}_i$

---

#### NOTE

**Position<sub>i</sub> (i=1..N) are the offset values relative to the scanning positioner start position that is defined in the PositionerCompensatedPCOPrepare().**

**The function works only when the CIE08 compensated PCO mode configuration is enabled (*system.ini*: CIE08CompensatedPCOMode = Enabled).**

**This function can be used only with a position encoder (“AquadB” or “AnalogInterpolated”), otherwise ERR\_UNCOMPATIBLE (-24) error is returned.**

---

### **Prototype**

```
int PositionerCompensatedPCOFromFile(
    int SocketID,
    char * PositionerName,
    char * DataFileName
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
DataFileName	char *	Data file name.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Incorrect file name, bad section name, or not available in this configuration (check hardware or configuration).
- -61: Error file corrupt or file doesn't exist.
- -115: Function is not supported by current hardware
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### **6.2.1.155 PositionerCompensatedPCOLoadToMemory**

#### **Name**

**PositionerCompensatedPCOLoadToMemory**– Appends firing positions to controller memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check the positioner type: (-8)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the CIE board supports this function: (-115)
- Check CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)
- Check data lines (must be  $\text{Position}_i > \text{Position}_{i-1}$ ,  $\text{Width}_i < \text{Position}_{i+1} - \text{Position}_i$ ): (-122)

#### **Description**

This function appends firing positions to controller’s memory from *DataLines* parameter.

To reset the controller’s memory, the *PositionerCompensatedPCOMemoryReset()* function is provided.

The data line format must be :

Position<sub>i</sub><Space or Tabulation>Width<sub>i</sub><CRLF, LF or ;>

#### ***Example :***

Position<sub>1</sub>Width<sub>1</sub>

Position<sub>2</sub>Width<sub>2</sub>

... ...

Position<sub>N</sub>Width<sub>N</sub>

Or :

Position<sub>1</sub>Width<sub>1</sub>;Position<sub>2</sub>Width<sub>2</sub>; ...;Position<sub>N</sub>Width<sub>N</sub>

Data conditions:  $\text{Position}_i > \text{Position}_{i-1}$ ,  $\text{Width}_i < \text{Position}_{i+1} - \text{Position}_i$

Example: Send PositionerCompensatedLoadToMemory (XY.X,0 0.1;1 0.1;2 0.1;3 0.1)

#### **NOTE**

**Position<sub>i</sub> (i=1..N) are the offset values relative to the scanning positioner start position that is defined in the PositionerCompensatedPCOPrepare().**

**The function works only when the CIE08 compensated PCO mode configuration is enabled (*system.ini*: CIE08CompensatedPCOMode = Enabled).**

**This function can be used only with a position encoder (“AquadB” or “AnalogInterpolated”), otherwise ERR\_UNCOMPATIBLE (-24) error is returned.**

### **Prototype**

```
int PositionerCompensatedPCOLoadToMemory(
    int SocketID,
    char * PositionerName,
    char * DataLine
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
DataLine	char *	Some data lines.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Incorrect file name, bad section name, or not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### **6.2.1.156 PositionerCompensatedPCOMemoryReset**

#### **Name**

**PositionerCompensatedPCOMemoryReset** – Resets CIE08 compensated PCO data memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check the positioner type: (-8)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the CIE board supports this function: (-115)
- Check CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)

#### **Description**

This function resets the CIE08 compensated PCO data memory. This function is useful to remove the data that was previously entered with the *PositionerCompensatedPCOLoadToMemory()* function.

---

#### **NOTE**

**The function works only when the CIE08 compensated PCO mode configuration is enabled (*system.ini*: CIE08CompensatedPCOMode = Enabled).**

**This function can be used only with a position encoder (“AquadB” or “AnalogInterpolated”), otherwise ERR\_UNCOMPATIBLE (-24) error is returned.**

---

#### **Prototype**

```
int PositionerCompensatedPCOMemoryReset(
    int SocketID,
    char * PositionerName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Incorrect file name, bad section name, or not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware
- -121: Function is not allowed due to configuration disabled.

### **6.2.1.157 PositionerCompensatedPCOPrepare**

#### **Name**

**PositionerCompensatedPCOPrepare** – Prepares data for CIE08 compensated PCO pulses generation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)
- Check the positioner type: (-8)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the CIE board supports this function: (-115)
- Check CIE08CompensatedPCOMode = Enabled (*system.ini*):  
ERR\_NOT\_ALLOWED\_MODE\_DISABLED (-121)
- Check data have been set, loaded or read from file to buffer (DataNumber must > 0 and < CIE08CompensatedPCOMaximumDataNumber (*system.ini*)):  
ERR\_CHECK\_DATA\_INCORRECT (-122)
- Check scanning zone exceed stage travel limits: (-35)
- Check input parameter values (ScanDirection value must equal to 1 (*positive direction*) or -1 (*negative direction*)): (-17)

#### **Description**

This function calculates the firing for absolute positions, *in user's coordinate system* and converts them to firing absolute raw PCO positions, *in encoder's coordinate system*.

When mappings are enabled, the correction between user's coordinate system position and raw encoder position will be different at each location. For this reason, the prepare function must know the location (*positions of all positioners in the scanning group*) where the scan will be done.

This function must be called before the use of *PositionerCompensatedPCOEnable()* function.

Parameters :

- ScanDirection: Scan direction,(value: 1 (*positive*) or -1 (*negative*)).
- StartPosition1: Group 1<sup>st</sup> positioner start position.
- StartPosition2: Group 2<sup>nd</sup> positioner start position
- StartPosition3: Group 3<sup>rd</sup> positioner start position
- ..... etc .....

---

#### **NOTE**

**The function works only when the CIE08 compensated PCO mode configuration is enabled (*system.ini*: CIE08CompensatedPCOMode = Enabled)..**

**This function can be used only with a position encoder (“AquadB” or “AnalogInterpolated”), otherwise ERR\_UNCOMPATIBLE (-24) error is returned.**

---

**Prototype**

```
int PositionerCompensatedPCOPPrepare(
    int SocketID,
    char * PositionerName
    int ScanDirection,
    double StartPosition1,
    double StartPosition2,
    double StartPosition3,
    ...
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
ScanDirection	int	Scan direction (1 or -1).
StartPosition1	double	Group 1 <sup>st</sup> positioner start position (units).
StartPosition2	double	Group 2 <sup>nd</sup> positioner start position (units).
StartPosition3	double	Group 3 <sup>rd</sup> positioner start position (units).
....		

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect
- -18: Positioner Name doesn't exist or unknown command.
- -24: Incorrect file name, bad section name, or not available in this configuration (check hardware or configuration).
- -35: Position is outside of travel limits.
- -115: Function is not supported by current hardware.
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### **6.2.1.158 PositionerCompensatedPCOSet**

#### **Name**

**PositionerCompensatedPCOSet** – Calculates a set of evenly spaced firing positions to the controller memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Start <Stop, Distance > 0, Width > 0, Width < Distance, Width < Stop-Start: (-17)
- Check the positioner name: (-18)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the CIE board supports this function: (-115)
- Check CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)
- Check data number NData = integer((Stop-Start)/Distance) + 1, if NData >1000000: (-122)

#### **Description**

This function calculates a set of evenly spaced firing positions to the controller memory.

Parameters :

- Start: Relative distance to the start position where PCO pulses start.
- Stop: Relative distance to the start position where PCO pulses stop.
- Distance: Step of pulses (distance between two consecutive pulses)
- Width: Width of the pulse enable signal at the firing positions.

Example: Send PositionerCompensatedPCOSet (XY.X, 0, 3, 1, 0.1)

#### **NOTE**

**Start and Stop are the offset values relative to the scanning positioner start position that is defined in the PositionerCompensatedPCOPrepare().**

**The function works only when the CIE08 compensated PCO mode configuration is enabled (*system.ini*: CIE08CompensatedPCOMode = Enabled).**

**This function can be used only with a position encoder (“AquadB” or “AnalogInterpolated”), otherwise ERR\_UNCOMPATIBLE (-24) error is returned.**

#### **Prototype**

```
int PositionerCompensatedPCOSet (int SocketID, char PositionerName[250] , double Start, double Stop, double Distance, double Width)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function
PositionerName	char *	Positioner name
Start	double	Start position (units)
Stop	double	Stop position (units)
Distance	double	Distance between two consecutive pulses (units)
Width	double	Width of pulse enable signal (units)

### **Output parameter**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner name doesn't exist or unknown command.
- -24: Incorrect file name, bad section name, or not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware
- -121: Function is not allowed due to configuration disabled.
- -122: Data incorrect (wrong value, wrong format, wrong order or nonexistent).

### **6.2.1.159 PositionerCompensationDisturbanceDisable**

#### **Name**

**PositionerCompensationDisturbanceDisable** – Disables disturbance compensation for selected direction (DisabledDirection = Both, Positive or Negative)

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the filter number: (-17)

#### **Description**

This function disables the compensation of disturbance for the selected direction (Both, Positive or Negative).

#### **Prototype**

```
int PositionerCompensationDisturbanceDisable(
```

```
    int SocketID,  
    char *PositionerName,  
    char *DisabledDirection  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
DisabledDirection	char*	“Both”, “Positive” or “Negative”

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### 6.2.1.160 PositionerCompensationDisturbanceEnable

#### Name

**PositionerCompensationDisturbanceEnable**— Enables disturbance compensation for selected direction (EnabledDirection = Both, Positive or Negative)

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the filter number: -17
- Check if a disturbance compensation file is loaded: -22

#### Description

This function enables the compensation of disturbance for the selected direction (Both, Positive or Negative). The disturbance compensation can be enabled only if a file was loaded (refer to API PositionerCompensationDisturbanceFileLoad ).

#### Prototype

```
int PositionerCompensationDisturbanceEnable(
```

```
    int SocketID,  
    char *PositionerName,  
    char *EnabledDirection  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
EnabledDirection	char*	“Both”, “Positive” or “Negative”

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action

### **6.2.1.161 PositionerCompensationDisturbanceFileLoad**

#### **Name**

**PositionerCompensationDisturbanceFileLoad** – Loads file to compensate disturbance in requested direction (Positive or Negative).

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the filter number: (-17)
- Check if the disturbance compensation feature is allowed in the firmware: -205
- Check if the disturbance compensation is enabled in the requested direction: -22

#### **Description**

This function loads a file that contains the compensation of disturbance in one direction. The file cannot be loaded if the disturbance compensation in the requested direction is already enabled. In this case, it can be disabled with the PositionerCompensationDisturbanceDisable API.

#### **Prototype**

```
int PositionerCompensationDisturbanceFileLoad(
    int SocketID,
    char *PositionerName,
    char *Direction,
    char *FileName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
Direction	char*	“Positive” or “Negative”.
FileName	char *	File name located in \Config folder.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action
- -61: Error file corrupt or file doesn't exist
- -205: Not enable in your configuration

### **6.2.1.162 PositionerCompensationDisturbanceStatusGet**

#### **Name**

**PositionerCompensationDisturbanceStatusGet** – Gets status of disturbance compensation in both directions.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check the filter number: (-17)

#### **Description**

This function returns the “Enabled” status of the disturbance compensation in both positive and negative directions.

#### **Prototype**

```
int PositionerCompensationDisturbanceStatusGet(  
    int SocketID,  
    char *PositionerName,  
    char * PositiveCompensationEnabledStatus,  
    char * NegativeCompensationEnabledStatus  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

PositiveCompensationEnabledStatus	char *	Current “Enabled” status in the positive direction
NegativeCompensationEnabledStatus	char *	Current “Enabled” status in the negative direction

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.163 PositionerCompensationDualLoopNotchFilterGet**

#### **Name**

**PositionerCompensationDualLoopNotchFilterGet** – Returns the notch filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check the notch filter number: (-17)
- Check the positioner name: (-18)
- Check dual corrector is enabled: (-205)

#### **Description**

This function returns the parameters defined for the selected notch filter in dual loop.

Notch filters parameters:

- NotchFrequency.
- NotchBandwidth.
- NotchGain.

#### **Prototype**

```
int PositionerCompensationDualLoopNotchFilterGet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double * NotchFrequency,
    double * NotchBandwidth,
    double * NotchGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter.

#### **Output parameters**

NotchFrequency	double *	Frequency (Hertz) for notch filter.
NotchBandwidth	double *	Band width (Hertz) for notch filter.
NotchGain	double *	Gain for notch filter.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -205: Not enabled in your configuration.

### **6.2.1.164 PositionerCompensationDualLoopNotchFilterSet**

#### **Name**

**PositionerCompensationDualLoopNotchFilterSet** – Sets the notch mode filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
  - NotchFrequency  $\in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right]$  with CorrectorPeriod = 0.0001 s  
(10 kHz) = > [0 : 5000]
  - NotchBandwidth with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
  - NotchGain  $\in [0 : 100]$
- Check the positioner name: (-18)

#### **Description**

This function configures the parameters defined for selected notch filter in dual loop. If the “NotchFrequency” value is NULL or the “NotchGain” value is NULL then the notch filter is not activated.

Notch filters parameters:

- NotchFrequency.
- NotchBandwidth.
- NotchGain.

#### **Prototype**

```
int PositionerCompensationDualLoopNotchFilterSet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double NotchFrequency,
    double NotchBandwith,
    double NotchGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter.
NotchFrequency	double	Frequency (Hertz) for notch filter.
NotchBandwith	double	Band width (Hertz) for notch filter.
NotchGain	double	Gain for notch filter.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.165 PositionerCompensationDualLoopPhaseCorrectionFilterGet**

#### **Name**

**PositionerCompensationDualLoopPhaseCorrectionFilterGet** – Returns the phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check the notch filter number: (-17)
- Check the positioner name: (-18)
- Check dual corrector is enabled: (-205)

#### **Description**

This function returns the system compensation parameters defined for selected phase correction filter in dual loop.

Phase correction filter parameters:

- PhaseCorrectionFn.
- PhaseCorrectionFd.
- PhaseCorrectionGain.

#### **Prototype**

```
int PositionerCompensationDualLoopPhaseCorrectionFilterGet(
    int SocketID,
    char * PositionerName,
    int PhaseCorrectionFilterNumber,
    double * PhaseCorrectionFn,
    double * PhaseCorrectionFd,
    double * PhaseCorrectionGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PhaseCorrectionFilterNumber	int	Number of the selected selected phase correction filter.

#### **Output parameters**

PhaseCorrectionFn	double *	Numerator frequency (Hertz) for phase correction filter.
PhaseCorrectionFd	double *	Denominator frequency (Hertz) for phase correction filter.
PhaseCorrectionGain	double *	Gain for phase correction filter.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -205: Not enabled in your configuration.

### **6.2.1.166 PositionerCompensationDualLoopPhaseCorrectionFilterSet**

#### **Name**

**PositionerCompensationDualLoopPhaseCorrectionFilterSet** – Sets the notch mode filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
  - PhaseCorrectionFn with CorrectorPeriod = 0.0001 s (10 kHz) =>  $[0 : 5000]$
  - PhaseCorrectionFd with CorrectorPeriod = 0.0001 s (10 kHz) =>  $[0 : 5000]$
  - PhaseCorrectionGain  $\geq 0$
- Check the positioner name: (-18)

#### **Description**

This function configures the parameters defined for the selected phase correction filter in dual loop. If the “PhaseCorrectionFn” value = 0 or the “PhaseCorrectionFd” value = 0 then the phase correction filter is not activated.

Phase correction filter parameters:

- PhaseCorrectionFn.
- PhaseCorrectionFd.
- PhaseCorrectionGain.

#### **Prototype**

```
int PositionerCompensationDualLoopNotchFilterSet(
    int SocketID,
    char * PositionerName,
    int PhaseCorrectionFilterNumber,
    double PhaseCorrectionFn,
    double PhaseCorrectionFd,
    double PhaseCorrectionGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PhaseCorrectionFilterNumber	int	Number of the selected selected phase correction filter.
PhaseCorrectionFn	double	Numerator frequency (Hertz) for phase correction filter.
PhaseCorrectionFd	double	Denominator frequency (Hertz) for phase correction filter.
PhaseCorrectionGain	double	Gain for phase correction filter.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### 6.2.1.167 **PositionerCompensationEncoderNotchFilterGet**

#### Name

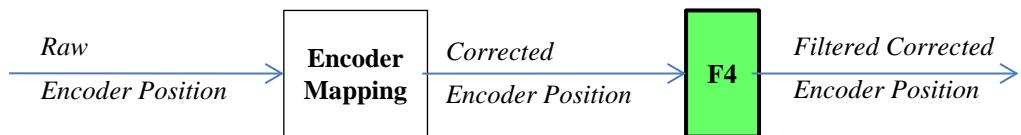
**PositionerCompensationEncoderNotchFilterGet** – Returns Encoder compensation notch filter parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check parameter values: (-17)
- Notch Frequency number  $\in [1:5]$

#### Description

This function returns parameters defined for the *[Dual]EncoderFilter* frequency notch filter from the encoder compensation (F4 compensation block) configured in encoder.



Encoder Notch filters parameters:

- NotchFrequency (Hertz).
- NotchBandwidth (Hertz).
- NotchGain.

#### Prototype

```
int PositionerCompensationEncoderNotchFilterGet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double * NotchFrequency,
    double * NotchBandwidth,
    double * NotchGain
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter (1 to 5).

#### Output parameters

NotchFrequency	double *	Frequency (Hertz) for Encoder Notch filter.
NotchBandwidth	double *	Band width (Hertz) for Encoder Notch filter.
NotchGain	double *	Gain for Encoder Notch filter.

**Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.168 PositionerCompensationEncoderNotchFilterSet

#### Name

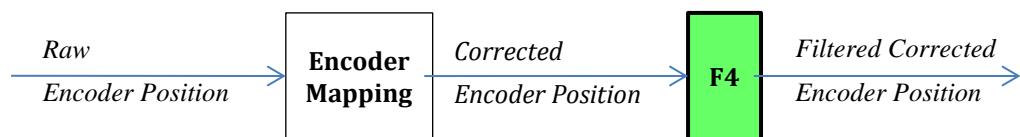
**PositionerCompensationEncoderNotchFilterSet** – Sets Encoder compensation notch filters parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check parameter values: (-17)
  - Notch Frequency number  $\in [1:5]$
  - NotchFrequency  $\in \left[0: \frac{0.5}{\text{CorrectorPeriod}}\right]$  with CorrectorPeriod = 0.0001 s  
(10 kHz)  $\Rightarrow [0:5000]$
  - NotchBandwidth  $\in \left[0: \frac{0.5}{\text{CorrectorPeriod}}\right]$  with CorrectorPeriod = 0.0001 s  
(10 kHz)  $\Rightarrow [0:5000]$
  - NotchGain  $\in [0:100]$

#### Description

This functions sets the selected *[Dual]EncoderFilter* frequency notch filter parameters from the encoder compensation (F4 compensation block) configured in encoder.



Encoder Notch filters parameters:

- NotchFrequency (Hertz).
- NotchBandwidth (Hertz).
- NotchGain.

#### Prototype

```

int PositionerCompensationEncoderNotchFilterSet(
    int SocketID,
    char * FullPositionerName,
    int NotchFrequencyNumber,
    double NotchFrequency,
    double NotchBandwidth,
    double NotchGain
)
  
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter (1 to 5).
NotchFrequency	double	Frequency (Hertz) for Encoder Notch filter.
NotchBandwidth	double	Band width (Hertz) for Encoder Notch filter.
NotchGain	double	Gain for Encoder Notch filter.

**Output parameters**

None.

**Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.169 PositionerCompensationFrequencyNotchsGet

#### Name

**PositionerCompensationFrequencyNotchsGet** – Gets pre-feedforward compensation notch filters parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: (-8)

#### Description

This function returns the *CompensationSystemPreFeedForward* frequency notch filters parameters. These notch filters allow the user to reduce external perturbations such as base motion or floor vibrations. Note that the *CompensationSystemPreFeedForward* feature is available for all corrector types (acceleration, velocity, voltage or position) functioning *in closed loop configuration*.

- NotchFrequency1
- NotchsBandwidth1
- NotchsGain1
- NotchFrequency2
- NotchsBandwidth2
- NotchsGain2
- NotchFrequency3
- NotchsBandwidth3
- NotchsGain3

#### Prototype

```
int PositionerCompensationFrequencyNotchsGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * NotchFrequency1,  
    double * NotchBandwidth1,  
    double * NotchGain1,  
    double * NotchFrequency2,  
    double * NotchBandwidth2,  
    double * NotchGain2,  
    double * NotchFrequency3,  
    double * NotchBandwidth3,  
    double * NotchGain3  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

### **Output parameters**

NotchFrequency1	double *	Notch frequency for filter #1 (Hz).
NotchBandwidth1	double *	Notch bandwidth for filter #1 (Hz).
NotchGain1	double *	Notch gain for filter #1.
NotchFrequency2	double *	Notch frequency for filter #2 (Hz).
NotchBandwidth2	double *	Notch bandwidth for filter #2 (Hz).
NotchGain2	double *	Notch gain for filter #2.
NotchFrequency3	double *	Notch frequency for filter #3 (Hz).
NotchBandwidth3	double *	Notch bandwidth for filter #3 (Hz).
NotchGain3	double *	Notch gain for filter #3.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.170 PositionerCompensationFrequencyNotchsSet

#### Name

**PositionerCompensationFrequencyNotchsSet** – Sets pre-feedforward compensation notch filters parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: ERR\_WRONG\_OBJECT\_TYPE (-8)
- Check parameter values: ERR\_PARAMETER\_OUT\_OF\_RANGE (-17)
  - NotchFrequency  $\in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$
  - NotchBandwidth  $\in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$

---

#### NOTE

Refer to *system.ref* file to get **CorrectorISRPeriod** value.

---

#### Description

This function sets the *CompensationSystemPreFeedForward* frequency notch filters parameters. These notch filters allow the user to reduce external perturbations such as base motion or floor vibrations. Note that the *CompensationSystemPreFeedForward* feature is available for all corrector types (acceleration, velocity, voltage or position) functioning in *closed loop configuration*.

NotchFrequency1  
NotchsBandwidth1  
NotchsGain1  
NotchFrequency2  
NotchsBandwidth2  
NotchsGain2  
NotchFrequency3  
NotchsBandwidth3  
NotchsGain3

**Prototype**

```
int PositionerCompensationFrequencyNotchsSet(
    int SocketID,
    char * FullPositionerName,
    double NotchFrequency1,
    double NotchBandwidth1,
    double NotchGain1,
    double NotchFrequency2,
    double NotchBandwidth2,
    double NotchGain2,
    double NotchFrequency3,
    double NotchBandwidth3,
    double NotchGain3
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
NotchFrequency1	double	Notch frequency for filter #1 (Hz).
NotchBandwidth1	double	Notch bandwidth for filter #1 (Hz).
NotchGain1	double	Notch gain for filter #1.
NotchFrequency2	double	Notch frequency for filter #2 (Hz).
NotchBandwidth2	double	Notch bandwidth for filter #2 (Hz).
NotchGain2	double	Notch gain for filter #2.
NotchFrequency3	double	Notch frequency for filter #3 (Hz).
NotchBandwidth3	double	Notch bandwidth for filter #3 (Hz).
NotchGain3	double	Notch gain for filter #3.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.171 PositionerCompensationLowPassTwoFilterGet**

#### **Name**

**PositionerCompensationLowPassTwoFilterGet** – Gets the post-feedforward compensation second order lowpass filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: (-8)

#### **Description**

This function returns the system compensation parameters defined for the post-feedforward compensation second order low-pass filter.

#### **Prototype**

```
int PositionerCompensationLowPassTwoFilterGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * CutOffFrequency  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

CutOffFrequency	double *	Second order filter cut-off frequency (Hertz).
-----------------	----------	--

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.172 PositionerCompensationLowPassTwoFilterSet**

#### **Name**

**PositionerCompensationLowPassTwoFilterSet** – Sets the post-feedforward compensation second order lowpass filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check correco type: (-8)
- Check parameter values: (-17)

- CutOffFrequency  $\in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$

#### **NOTE**

Refer to *system.ref* file to get CorrectorISRPeriod value.

#### **Description**

This function configures the parameters defined for the post-feedforward compensation second order low-pass filter.

#### **NOTE**

If the “CutOffFrequency” value = 0 then the second order low-pass filter is not activated.

#### **Prototype**

```
int PositionerCompensationLowPassTwoFilterSet(
    int SocketID,
    char * FullPositionerName,
    double CutOffFrequency
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
CutOffFrequency	double	Second order filter cut-off frequency (Hertz).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.173 PositionerCompensationNotchFilterGet**

#### **Name**

**PositionerCompensationNotchFilterGet** – Returns the notch filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check phase correction number: (-17)
- Check the positioner name: (-18)

#### **Description**

This function returns the parameters defined for the selected notch filter.

Notch filters parameters:

- NotchFrequency.
- NotchBandwidth.
- NotchGain.

#### **Prototype**

```
int PositionerCompensationNotchFilterGet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double * NotchFrequency,
    double * NotchBandwith,
    double * NotchGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter.

#### **Output parameters**

NotchFrequency	double *	Frequency (Hertz) for notch filter.
NotchBandwith	double *	Band width (Hertz) for notch filter.
NotchGain	double *	Gain for notch filter.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -205: Not enable in your configuration.

### **6.2.1.174 PositionerCompensationNotchFilterSet**

#### **Name**

**PositionerCompensationNotchFilterSet** – Sets the notch filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
  - NotchFrequency with CorrectorPeriod = 0.0001 s (10 kHz) =>  $[0 : 5000]$
  - NotchBandwidth with CorrectorPeriod = 0.0001 s (10 kHz) =>  $[0 : 5000]$
  - NotchGain  $\in [0 : 100]$
- Check the positioner name: (-18)
- Check compensation preFeed forward mode is enabled: (-24)
- Check the motion status (Motion status must be disable): (-134)

#### **Description**

This function configures the parameters defined for the selected notch filter. If the “NotchFrequency” value is NULL or the “NotchGain” value is NULL then the notch filter is not activated.

Notch filter parameters:

- NotchFrequency.
- NotchBandwidth.
- NotchGain.

#### **Prototype**

```
int PositionerCompensationNotchFilterSet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double NotchFrequency,
    double NotchBandwith,
    double NotchGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected phase correction filter.
NotchFrequency	double	Frequency (Hertz) for notch filter.
NotchBandwith	double	Band width (Hertz) for notch filter.
NotchGain	double	Gain for notch filter.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -205: Not enabled in your configuration.
- -134: Changing the loop status is allowed in DISABLE state only.

### **6.2.1.175 PositionerCompensationNotchModeFiltersGet**

#### **Name**

**PositionerCompensationNotchModeFiltersGet** – Gets the post-feedforward compensation notch mode filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: (-8)

#### **Description**

This functions returns the system compensation parameters defined for two post-feedforward compensation notch mode filters.

First notch mode filter parameters:

- NotchModeFr1.
- NotchModeFa1.
- NotchModeZr1.
- NotchModeZa1.

Second notch mode filter parameters:

- NotchModeFr2.
- NotchModeFa2.
- NotchModeZr2.
- NotchModeZa2.

#### **Prototype**

```
int PositionerCompensationNotchModeFiltersGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * NotchModeFr1,  
    double * NotchModeFa1,  
    double * NotchModeZr1,  
    double * NotchModeZa1,  
    double * NotchModeFr2,  
    double * NotchModeFa2,  
    double * NotchModeZr2,  
    double * NotchModeZa2  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

### **Output parameters**

NotchModeFr1	double *	Resonance frequency (Hertz) for notch mode filter #1.
NotchModeFa1	double *	Anti-resonance frequency (Hertz) for notch mode filter #1.
NotchModeZr1	double *	Resonance damping factor for notch mode filter #1.
NotchModeZa1	double *	Anti-resonance damping factor for notch mode filter #1.
NotchModeFr2	double *	Resonance frequency (Hertz) for notch mode filter #2.
NotchModeFa2	double *	Anti-resonance frequency (Hertz) for notch mode filter #2.
NotchModeZr2	double *	Resonance damping factor for notch filter #2.
NotchModeZa2	double *	Anti-resonance damping factor for notch mode filter #2.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.176 PositionerCompensationNotchModeFiltersSet

#### Name

**PositionerCompensationNotchModeFiltersSet** – Sets the post-feedforward compensation notch mode filter parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: (-8)
- Check parameter values: (-17)

$$\begin{aligned} \bullet \quad \text{NotchModeFr} &\in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right] \\ \bullet \quad \text{NotchModeFa} &\in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right] \end{aligned}$$

---

#### NOTE

Refer to *system.ref* file to get **CorrectorISRPeriod** value.

---

#### Description

This function configures the parameters defined for two post-feedforward compensation notch mode filters.

First notch mode filter parameters:

- NotchModeFr1.
- NotchModeFa1.
- NotchModeZr1.
- NotchModeZa1.

Second notch mode filter parameters:

- NotchModeFr2.
- NotchModeFa2.
- NotchModeZr2.
- NotchModeZa2.

---

#### NOTE

If the “NotchModeFr” value = 0 or the “NotchModeFa” value = 0, then the notch mode filter is not activated.

---

**Prototype**

```
int PositionerCompensationNotchModeFiltersSet(
    int SocketID,
    char * FullPositionerName,
    double NotchModeFr1,
    double NotchModeFa1,
    double NotchModeZr1,
    double NotchModeZa1,
    double NotchModeFr2,
    double NotchModeFa2,
    double NotchModeZr2,
    double NotchModeZa2
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
NotchModeFr1	double	Resonance frequency (Hertz) for notch mode filter #1.
NotchModeFa1	double	Anti-resonance frequency (Hertz) for notch mode filter #1.
NotchModeZr1	double	Resonance damping factor for notch mode filter #1.
NotchModeZa1	double	Anti-resonance damping factor for notch mode filter #1.
NotchModeFr2	double	Resonance frequency (Hertz) for notch mode filter #2.
NotchModeFa2	double	Anti-resonance frequency (Hertz) for notch mode filter #2.
NotchModeZr2	double	Resonance damping factor for notch mode filter #2.
NotchModeZa2	double	Anti-resonance damping factor for notch mode filter #2.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.177 PositionerCompensationPhaseCorrectionFilterGet**

#### **Name**

**PositionerCompensationPhaseCorrectionFilterGet** – Returns the phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check phase correction number: (-17)
- Check the positioner name: (-18)

#### **Description**

This function returns the system compensation parameters defined for selected phase correction filter.

Phase correction filter parameters:

- PhaseCorrectionFn.
- PhaseCorrectionFd.
- PhaseCorrectionGain.

#### **Prototype**

```
int PositionerCompensationPhaseCorrectionFilterGet(
    int SocketID,
    char * PositionerName,
    int PhaseCorrectionFilterNumber,
    double * PhaseCorrectionFn,
    double * PhaseCorrectionFd,
    double * PhaseCorrectionGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PhaseCorrectionFilterNumber	int	Number of the selected Notch Frequency filter.

#### **Output parameters**

PhaseCorrectionFn	double *	Frequency (Hertz) for notch filter.
PhaseCorrectionFd	double *	Band width (Hertz) for notch filter.
PhaseCorrectionGain	double *	Gain for notch filter.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -15: Wrong parameter type in the command string: int, short, int \* or short \* expected.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.178 PositionerCompensationPhaseCorrectionFilterSet**

#### **Name**

**PositionerCompensationPhaseCorrectionFilterSet** – Sets the notch filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
  - PhaseCorrectionFn with CorrectorPeriod = 0.0001 s (10 kHz) =>  $[0 : 5000]$
  - PhaseCorrectionFd with CorrectorPeriod = 0.0001 s (10 kHz) =>  $[0 : 5000]$
  - PhaseCorrectionGain  $\geq 0$
- Check the positioner name: (-18)
- Check the motion status (Motion status must be disable): (-134)

#### **Description**

This function configures the parameters defined for the selected phase correction filter. If the “PhaseCorrectionFn” value = 0 or the “PhaseCorrectionFd” value = 0 then the phase correction filter is not activated.

Phase correction filter parameters:

- PhaseCorrectionFn.
- PhaseCorrectionFd.
- PhaseCorrectionGain.

#### **Prototype**

```
int PositionerCompensationPhaseCorrectionFilterSet(
    int SocketID,
    char * PositionerName,
    int PhaseCorrectionFilterNumber,
    double PhaseCorrectionFn,
    double PhaseCorrectionFd,
    double PhaseCorrectionGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PhaseCorrectionFilterNumber	int	Number of the selected Notch Frequency filter.
PhaseCorrectionFn	double	Frequency (Hertz) for notch filter.
PhaseCorrectionFd	double	Band width (Hertz) for notch filter.
PhaseCorrectionGain	double	Gain for notch filter.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -205: Not enabled in your configuration.
- -134: Changing the loop status is allowed in DISABLE state only.

### **6.2.1.179 PositionerCompensationPhaseCorrectionFiltersGet**

#### **Name**

**PositionerCompensationPhaseCorrectionFiltersGet** – Returns the post-feedforward compensation phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: (-8)

#### **Description**

This function returns the system compensation parameters defined for two post-feedforward compensation phase correction filters.

First phase correction filter parameters:

- PhaseCorrectionFn1.
- PhaseCorrectionFd1.
- PhaseCorrectionGain1.

Second phase correction filter parameters:

- PhaseCorrectionFn2.
- PhaseCorrectionFd2.
- PhaseCorrectionGain2.

#### **Prototype**

```
int PositionerCompensationPhaseCorrectionFiltersGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * PhaseCorrectionFn1,  
    double * PhaseCorrectionFd1,  
    double * PhaseCorrectionGain1,  
    double * PhaseCorrectionFn2,  
    double * PhaseCorrectionFd2,  
    double * PhaseCorrectionGain2  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

### **Output parameters**

PhaseCorrectionFn1	double *	Numerator frequency (Hertz) for phase correction filter #1.
PhaseCorrectionFd1	double *	Denominator frequency (Hertz) for phase correction filter #1.
PhaseCorrectionGain1	double *	Gain for phase correction filter #1.
PhaseCorrectionFn2	double *	Numerator frequency (Hertz) for phase correction filter #2.
PhaseCorrectionFd2	double *	Denominator frequency (Hertz) for phase correction filter #2.
PhaseCorrectionGain2	double *	Gain for phase correction filter #2.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.180 PositionerCompensationPhaseCorrectionFiltersSet**

#### **Name**

**PositionerCompensationPhaseCorrectionFiltersSet** – Sets the post-feedforward compensation phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: (-8)
- Check parameter values: (-17)
  - PhaseCorrectionFn  $\in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$
  - PhaseCorrectionFd

#### **NOTE**

Refer to *system.ref* file to get CorrectorISRPeriod value.

#### **Description**

This functions configures the parameters defined for two post-feedforward compensation phase correction filters.

First phase correction filter parameters:

- PhaseCorrectionFn1.
- PhaseCorrectionFd1.
- PhaseCorrectionGain1.

Second phase correction filter parameters:

- PhaseCorrectionFn2.
- PhaseCorrectionFd2.
- PhaseCorrectionGain2.

#### **NOTE**

If the “PhaseCorrectionFn” value = 0 or the “PhaseCorrectionFd” value = 0 then the phase correction filter is not activated.

#### **Prototype**

```
int PositionerCompensationPhaseCorrectionFiltersSet(
    int SocketID,
    char * FullPositionerName,
    double PhaseCorrectionFn1,
    double PhaseCorrectionFd1,
    double PhaseCorrectionGain1,
    double PhaseCorrectionFn2,
    double PhaseCorrectionFd2,
    double PhaseCorrectionGain2
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
PhaseCorrectionFn1	double	Numerator frequency (Hertz) for phase correction filter #1.
PhaseCorrectionFd1	double	Denominator frequency (Hertz) for phase correction filter #1.
PhaseCorrectionGain1	double	Gain for phase correction filter #1.
PhaseCorrectionFn2	double	Numerator frequency (Hertz) for phase correction filter #2.
PhaseCorrectionFd2	double	Denominator frequency (Hertz) for phase correction filter #2.
PhaseCorrectionGain2	double	Gain for phase correction filter #2.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.181 PositionerCompensationPositionFilterGet**

#### **Name**

**PositionerCompensationPositionFilterGet** – Returns the Position filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the filter number: (-17)

#### **Description**

This function returns the parameters defined for the selected position filter.

Position filters parameters:

- Frequency.
- DampingFactor.

#### **Prototype**

```
int PositionerCompensationPositionFilterGet(
```

```
    int SocketID,  
    char * PositionerName,  
    int PositionFilterNumber,  
    double * Frequency,  
    double * DampingFactor  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PositionFilterNumber	int	Number of the selected Position. Frequency filter.

#### **Output parameters**

Frequency	double *	Frequency (Hertz) for notch filter.
DampingFactor	double *	Damping factor for Position filter.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.182 PositionerCompensationPositionFilterSet**

#### **Name**

**PositionerCompensationPositionFilterSet**– Sets the Position filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the filter number: (-17)

#### **Description**

This function configures the parameters defined for selected Position filter. If the “Frequency” value is NULL then the Position filter is not activated.

Position filters parameters:

- Frequency.
- DampingFactor.

#### **Prototype**

```
int PositionerCompensationPositionFilterSet(
    int SocketID,
    char * PositionerName,
    int PositionFilterNumber,
    double Frequency,
    double DampingFactor
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PositionFilterNumber	int	Number of the selected Position. Frequency filter.
Frequency	double	Frequency (Hertz) for notch filter.
DampingFactor	double	Damping factor for Position filter.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.183 PositionerCompensationPostExcitationFrequencyNotchFilterGet**

#### **Name**

**PositionerCompensationPostExcitationFrequencyNotchFilterGet** – Returns Notch filter parameters from F3 compensation block.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Notch Frequency number (1 to 10): (-17)

#### **Description**

This function returns the parameters defined for the selected notch filter from the post excitation compensation (F3 compensation block) configured in the current corrector (Option0, Option1 or Option2).

Notch filters parameters:

- NotchFrequency (Hertz).
- NotchBandwidth (Hertz).
- NotchGain.

#### **Prototype**

```
int PositionerCompensationPostExcitationFrequencyNotchFilterGet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double * NotchFrequency,
    double * NotchBandwith,
    double * NotchGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter (1 to 10).

#### **Output parameters**

NotchFrequency	double *	Frequency (Hertz) for notch filter.
NotchBandwith	double *	Band width (Hertz) for notch filter.
NotchGain	double *	Gain for notch filter.

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -17: Parameter out of range or incorrect.

### **6.2.1.184 PositionerCompensationPostExcitationFrequencyNotchFilterSet**

#### **Name**

**PositionerCompensationPostExcitationFrequencyNotchFilterSet** – Sets Notch filter parameters from F3 compensation block.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Notch Frequency number (1 to 10): (-17)
- Check parameter values: (-17)
  - NotchFrequency  $\in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right]$  with CorrectorPeriod = 0.0001 s  
(10 kHz)  $= > [0 : 5000]$
  - NotchBandwidth  $\in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right]$  with CorrectorPeriod = 0.0001 s  
(10 kHz)  $= > [0 : 5000]$
  - NotchGain  $\in [0 : 100]$

#### **Description**

This function configures the parameters defined for selected Notch filter from the post excitation compensation (F3 block) configured in the current corrector (Option0, Option1 or Option2).

If the “NotchFrequency” value is NULL or the “NotchGain” value is NULL then the notch filter is not activated.

Notch filter parameters:

- NotchFrequency (Hertz).
- NotchBandwidth (Hertz).
- NotchGain.

#### **Prototype**

```
int PositionerCompensationPostExcitationFrequencyNotchFilterSet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double NotchFrequency,
    double NotchBandwidth,
    double NotchGain
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter (1 to 10).
NotchFrequency	double	Frequency (Hertz) for notch filter.
NotchBandwidth	double	Band width (Hertz) for notch filter.
NotchGain	double	Gain for notch filter.

**Output parameters**

None.

**Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -17: Parameter out of range or incorrect.

### **6.2.1.185 PositionerCompensationPostExcitationLowPassFilterGet**

#### **Name**

**PositionerCompensationPostExcitationLowPassFilterGet** – Returns the phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check the positioner name: (-18)

#### **Description**

This function returns the system compensation parameters defined for the second order low-pass filter.

#### **Prototype**

```
int PositionerCompensationPostExcitationLowPassFilterGet(  
    int SocketID,  
    char * PositionerName,  
    double * CutOffFrequency  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

CutOffFrequency	double *	Second order filter cut-off frequency (Hertz).
-----------------	----------	--

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.186 PositionerCompensationPostExcitationLowPassFilterSet**

#### **Name**

**PositionerCompensationPostExcitationLowPassFilterSet** – Sets second order low-pass filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
  - CutOffFrequency with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
- Check the positioner name: (-18)
- Check compensation post excitation mode is enabled: (-24)
- Check the motion status (Motion status must be disable): (-134)

#### **Description**

This function configures the parameters defined for the second order low-pass filter. If the “CutOffFrequency” value = 0 then the second order low-pass filter is not activated.

#### **Prototype**

```
int PositionerCompensationPostExcitationLowPassFilterSet(
    int SocketID,
    char * PositionerName,
    double CutOffFrequency
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
CutOffFrequency	double	Second order filter cut-off frequency (Hertz).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
- -134: Changing the loop status is allowed in DISABLE state only.

### **6.2.1.187 PositionerCompensationPostExcitationNotchModeFilterGet**

#### **Name**

**PositionerCompensationPostExcitationNotchModeFilterGet** – Returns the notch mode filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check phase correction number: (-17)
- Check the positioner name: (-18)

#### **Description**

This functions returns the system compensation parameters defined for two notch mode filters.

Notch mode filter parameters:

- NotchModeFr: Resonance frequency (Hertz)
- NotchModeFa: Anti-resonance frequency (Hertz)
- NotchModeZr: Resonance damping factor.
- NotchModeZa: Anti-resonance damping factor.

#### **Prototype**

```
int PositionerCompensationPostExcitationNotchModeFilterGet(
    int SocketID,
    char * PositionerName,
    int NotchModeNumber,
    double * NotchModeFr,
    double * NotchModeFa,
    double * NotchModeZr,
    double * NotchModeZa
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchModeNumber	int	Number of the selected notch mode filter.

#### **Output parameters**

NotchModeFr	double *	Resonance frequency (Hertz) for notch mode filter.
NotchModeFa	double *	Anti-resonance frequency (Hertz) for notch mode filter.
NotchModeZr	double *	Resonance damping factor for notch mode filter.
NotchModeZa	double *	Anti-resonance damping factor for notch mode filter.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.188 PositionerCompensationPostExcitationNotchModeFilterSet**

#### **Name**

**PositionerCompensationPostExcitationNotchModeFilterSet** – Sets the notch mode filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
  - NotchModeFr with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
  - NotchModeFa with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
- Check the positioner name: (-18)
- Check compensation post excitation mode is enabled: (-24)
- Check the motion status (Motion status must be disable): (-134)

#### **Description**

This functions returns the system compensation parameters defined for two notch mode filters.

Notch mode filter parameters:

- NotchModeFr: Resonance frequency (Hertz)
- NotchModeFa: Anti-resonance frequency (Hertz)
- NotchModeZr: Resonance damping factor.
- NotchModeZa: Anti-resonance damping factor.

#### **Prototype**

```
int PositionerCompensationPostExcitationNotchModeFilterSet(  
    int SocketID,  
    char * PositionerName,  
    int NotchModeNumber,  
    double * NotchModeFr,  
    double * NotchModeFa,  
    double * NotchModeZr,  
    double * NotchModeZa  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchModeNumber	int	Number of the selected notch mode filter.
NotchModeFr	double	Resonance frequency (Hertz) for notch mode filter.
NotchModeFa	double	Anti-resonance frequency (Hertz) for notch mode filter.
NotchModeZr	double	Resonance damping factor for notch mode filter.
NotchModeZa	double	Anti-resonance damping factor for notch mode filter.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
- -134: Changing the loop status is allowed in DISABLE state only.

### **6.2.1.189 PositionerCompensationPostExcitationPhaseCorrectionFilterGet**

#### **Name**

**PositionerCompensationPostExcitationPhaseCorrectionFilterGet** – Returns the phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check the phase correction filter number: (-17)
- Check the positioner name: (-18)

#### **Description**

This function returns the system compensation parameters defined for selected phase correction filter.

Phase correction filter parameters:

- PhaseCorrectionFn.
- PhaseCorrectionFd.
- PhaseCorrectionGain.

#### **Prototype**

```
int PositionerCompensationPostExcitationPhaseCorrectionFilterGet(  
    int SocketID,  
    char * PositionerName,  
    int PhaseCorrectionFilterNumber,  
    double * PhaseCorrectionFn,  
    double * PhaseCorrectionFd,  
    double * PhaseCorrectionGain  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PhaseCorrectionFilterNumber	int	Number of the selected phase correction filter.

#### **Output parameters**

PhaseCorrectionFn	double *	Numerator frequency (Hertz) for phase correction filter.
PhaseCorrectionFd	double *	Denominator frequency (Hertz) for phase correction filter.
PhaseCorrectionGain	double *	Gain for phase correction filter.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.190 PositionerCompensationPostExcitationPhaseCorrectionFilterSet**

#### **Name**

**PositionerCompensationPostExcitationPhaseCorrectionFilterSet** – Sets the phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
  - PhaseCorrectionFn with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
  - PhaseCorrectionFd with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
  - PhaseCorrectionGain  $\geq 0$
- Check the positioner name: (-18)
- Check compensation post excitation mode is enabled: (-24)
- Check the motion status (Motion status must be disable): (-134)

#### **Description**

This function configures the parameters defined for the selected phase correction filter. If the “PhaseCorrectionFn” value = 0 or the “PhaseCorrectionFd” value = 0 then the phase correction filter is not activated.

Phase correction filter parameters:

- PhaseCorrectionFn.
- PhaseCorrectionFd.
- PhaseCorrectionGain.

#### **Prototype**

```
int PositionerCompensationPostExcitationPhaseCorrectionFilterSet(  
    int SocketID,  
    char * PositionerName,  
    int PhaseCorrectionFilterNumber,  
    double * PhaseCorrectionFn,  
    double * PhaseCorrectionFd,  
    double * PhaseCorrectionGain  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PhaseCorrectionFilterNumber	int	Number of the selected phase correction filter.
PhaseCorrectionFn	double *	Numerator frequency (Hertz) for phase correction filter.
PhaseCorrectionFd	double *	Denominator frequency (Hertz) for phase correction filter.
PhaseCorrectionGain	double *	Gain for phase correction filter.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
- -134: Changing the loop status is allowed in DISABLE state only.

### **6.2.1.191 PositionerCompensationPreFeedForwardFrequencyNotchFilterGet**

#### **Name**

**PositionerCompensationPreFeedForwardFrequencyNotchFilterGet** – Returns the notch filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameter value: (-17)
- Check the positioner name: (-18)

#### **Description**

This function returns the parameters defined for the selected notch filter.

Notch filters parameters:

- UserNotchFrequency.
- UserNotchBandwidth.
- UserNotchGain.

#### **Prototype**

```
int PositionerCompensationPreFeedForwardFrequencyNotchFilterGet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double * NotchFrequency,
    double * NotchBandwith,
    double * NotchGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter.

#### **Output parameters**

NotchFrequency	double *	Frequency (Hertz) for notch filter.
NotchBandwith	double *	Band width (Hertz) for notch filter.
NotchGain	double *	Gain for notch filter.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.192 PositionerCompensationPreFeedForwardFrequencyNotchFilterSet**

#### **Name**

**PositionerCompensationPreFeedForwardFrequencyNotchFilterSet** – Sets the notch filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
  - NotchFrequency with CorrectorPeriod = 0.0001 s (10 kHz) => [0 : 5000]
  - NotchBandwidth with CorrectorPeriod = 0.0001 s (10 kHz) => [0 : 5000]
  - NotchGain
- Check the positioner name: (-18)
- Check compensation preFeed forward mode is enabled: (-24)
- Check the motion status (Motion status must be disable): (-134)

#### **Description**

This function configures the parameters defined for selected notch filter. If the “NotchFrequency” value is NULL or the “NotchGain” value is NULL then the notch filter is not activated.

Notch filters parameters:

- UserNotchFrequency.
- UserNotchBandwidth.
- UserNotchGain.

#### **Prototype**

```
int PositionerCompensationPreFeedForwardFrequencyNotchFilterSet(
    int SocketID,
    char * PositionerName,
    int NotchFrequencyNumber,
    double NotchFrequency,
    double NotchBandwith,
    double NotchGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
NotchFrequencyNumber	int	Number of the selected Notch Frequency filter.
NotchFrequency	double	Frequency (Hertz) for notch filter.
NotchBandwith	double	Band width (Hertz) for notch filter.
NotchGain	double	Gain for notch filter.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
- -134: Changing the loop status is allowed in DISABLE state only.

### **6.2.1.193 PositionerCompensationPreFeedForwardPhaseCorrectionFilterGet**

#### **Name**

**PositionerCompensationPreFeedForwardPhaseCorrectionFilterGet** – Returns the phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check phase correction number: (-17)
- Check the positioner name: (-18)

#### **Description**

This function returns the system compensation parameters defined for selected phase correction filter.

Phase correction filter parameters:

- PhaseCorrectionFn.
- PhaseCorrectionFd.
- PhaseCorrectionGain.

#### **Prototype**

```
int PositionerCompensationPreFeedForwardPhaseCorrectionFilterGet(
    int SocketID,
    char * PositionerName,
    int PhaseCorrectionFilterNumber,
    double * PhaseCorrectionFn,
    double * PhaseCorrectionFd,
    double * PhaseCorrectionGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PhaseCorrectionFilterNumber	int	Number of the selected phase correction filter.

#### **Output parameters**

PhaseCorrectionFn	double *	Numerator frequency (Hertz) for phase correction filter.
PhaseCorrectionFd	double *	Denominator frequency (Hertz) for phase correction filter.
PhaseCorrectionGain	double *	Gain for phase correction filter.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.194 PositionerCompensationPreFeedForwardPhaseCorrectionFilterSet**

#### **Name**

**PositionerCompensationPreFeedForwardPhaseCorrectionFilterSet** – Sets the phase correction filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters number: (-17)
  - PhaseCorrectionFn with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
  - PhaseCorrectionFd with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
  - PhaseCorrectionGain  $\geq 0$
- Check the positioner name: (-18)
- Check compensation preFeed forward mode is enabled: (-24)
- Check the motion status (Motion status must be disable): (-134)

#### **Description**

This function configures the parameters defined for the selected phase correction filter. If the “PhaseCorrectionFn” value = 0 or the “PhaseCorrectionFd” value = 0 then the phase correction filter is not activated.

Phase correction filter parameters:

- PhaseCorrectionFn.
- PhaseCorrectionFd.
- PhaseCorrectionGain.

#### **Prototype**

```
int PositionerCompensationPreFeedForwardPhaseCorrectionFilterSet(  
    int SocketID,  
    char * PositionerName,  
    int PhaseCorrectionFilterNumber,  
    double PhaseCorrectionFn,  
    double PhaseCorrectionFd,  
    double PhaseCorrectionGain  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PhaseCorrectionFilterNumber	int	Number of the selected phase correction filter.
PhaseCorrectionFn	double	Numerator frequency (Hertz) for phase correction filter.
PhaseCorrectionFd	double	Denominator frequency (Hertz) for phase correction filter.
PhaseCorrectionGain	double	Gain for phase correction filter.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -134: Changing the loop status is allowed in DISABLE state only.

### **6.2.1.195 PositionerCompensationPreFeedForwardSpatialNotchFilterGet**

#### **Name**

**PositionerCompensationPreFeedForwardSpatialNotchFilterGet** – Returns the spatial notch filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check spatial notch number: (-17)
- Check the positioner name: (-18)

#### **Description**

This function returns the parameters defined for the selected spatial notch filter.

Spatial notch filters parameters:

- SpatialNotchStep.
- SpatialNotchBandwidth.
- SpatialNotchGain.

#### **Prototype**

```
int PositionerCompensationPreFeedForwardSpatialNotchFilterGet(
    int SocketID,
    char * PositionerName,
    int SpatialNotchNumber,
    double * SpatialNotchStep,
    double * SpatialNotchBandwidth,
    double * SpatialNotchGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
SpatialNotchNumber	int	Number of the selected Spatial Notch Frequency filter.

#### **Output parameters**

SpatialNotchStep	double *	Step for spatial notch filter.
SpatialNotchBandwidth	double *	Band width (Hertz) for spatial notch filter.
SpatialNotchGain	double *	Gain for spatial notch filter.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.196 PositionerCompensationPreFeedForwardSpatialNotchFilterSet**

#### **Name**

**PositionerCompensationPreFeedForwardSpatialNotchFilterSet** – Sets the notch filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)

- If ( $\text{SpatialNotchStep} > 1.0\text{e-12}$ )

$$\frac{\text{MaximumVelocity}}{\text{SpatialNotchStep}} \in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right] \quad (\text{CorrectorPeriod: see system.ref})$$

$$\bullet \quad \text{SpatialNotchBandwidth} \in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right]$$

$$\bullet \quad \text{SpatialNotchGain} \in [0:100]$$

- Else

$$\bullet \quad \text{SpatialNotchStep} = 0$$

$$\bullet \quad \text{SpatialNotchBandwidth} = 0$$

$$\bullet \quad \text{SpatialNotchGain} = 1$$

- Check the positioner name: (-18)

- Check compensation preFeed forward mode is enabled: (-24)

- Check the motion status (Motion status must be disable): (-134)

#### **Description**

This function configures the parameters defined for selected spatial notch filter. If the “SpatialNotchStep” value is 0 then the spatial notch filter is not activated and the gain is setting to 1.

#### **Prototype**

```
int PositionerCompensationPreFeedForwardSpatialNotchFilterSet(
    int SocketID,
    char * PositionerName,
    int SpatialNotchNumber,
    double SpatialNotchStep,
    double SpatialNotchBandwidth,
    double SpatialNotchGain
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
SpatialNotchStep	double	Step for spatial notch filter.
SpatialNotchBandwidth	double	Band width (Hertz) for spatial notch filter.
SpatialNotchGain	double	Gain for spatial notch filter.
SpatialNotchNumber	int	Number of the selected Spatial Notch Frequency filter.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
- -134: Changing the loop status is allowed in DISABLE state only.

### **6.2.1.197 PositionerCompensationSpatialPeriodicNotchsGet**

#### **Name**

**PositionerCompensationSpatialPeriodicNotchsGet** – Returns pre-feedforward compensation spatial periodic filters parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: (-8)

#### **Description**

This functions returns the *CompensationSystemPreFeedForward* spatial periodic filters parameters. These filters reduce the spatial periodic perturbations coming from screw pitch or cogging.

Note that the *CompensationSystemPreFeedForward* feature is available for all corrector types (acceleration, velocity, voltage or position) functioning *in closed loop configuration*.

- SpatialNotchStep1.
- SpatialNotchsBandwidth1.
- SpatialNotchsGain1.
- SpatialNotchStep2.
- SpatialNotchsBandwidth2.
- SpatialNotchsGain2.
- SpatialNotchStep3.
- SpatialNotchsBandwidth3.
- SpatialNotchsGain3.

#### **Prototype**

```
int PositionerCompensationSpatialPeriodicNotchsGet(
    int SocketID,
    char * FullPositionerName,
    double * SpatialNotchStep1,
    double * SpatialNotchBandwidth1,
    double * SpatialNotchGain1,
    double * SpatialNotchStep2,
    double * SpatialNotchBandwidth2,
    double * SpatialNotchGain2,
    double * SpatialNotchStep3,
    double * SpatialNotchBandwidth3,
    double * SpatialNotchGain3
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

### **Output parameters**

SpatialNotchStep1	double *	Spatial periodic step for filter #1 (units).
SpatialNotchBandwidth1	double *	Spatial periodic bandwidth for filter #1 (Hz).
SpatialNotchGain1	double *	Spatial periodic gain for filter #1.
SpatialNotchStep2	double *	Spatial periodic step for filter #2 (units).
SpatialNotchBandwidth2	double *	Spatial periodic bandwidth for filter #2 (Hz).
SpatialNotchGain2	double *	Spatial periodic gain for filter #2.
SpatialNotchStep3	double *	Spatial periodic step for filter #3 (units).
SpatialNotchBandwidth3	double *	Spatial periodic bandwidth for filter #3 (Hz).
SpatialNotchGain3	double *	Spatial periodic gain for filter #3.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.198 PositionerCompensationSpatialPeriodicNotchsSet

#### Name

**PositionerCompensationSpatialPeriodicNotchsSet** – Sets pre-feedforward compensation spatial periodic filters parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check corrector type: (-8)
- Check parameter values: (-17)
  - SpatialNotchStep  $\in [0 : \text{MaximumVelocity} * \text{CorrectorISRPeriod}]$
  - SpatialNotchBandwidth  $\in \left[0 : \frac{0.5}{\text{CorrectorISRPeriod}}\right]$

---

#### NOTE

Refer to system.ref file to get **CorrectorISRPeriod** and stages.ini for **MaximumVelocity** values.

---

#### Description

This function sets the *CompensationSystemPreFeedForward* spatial periodic filters parameters. These filters reduce the spatial periodic perturbations coming from screw pitch or cogging.

Note that the *CompensationSystemPreFeedForward* feature is available for all corrector types (acceleration, velocity, voltage or position) functioning *in closed loop configuration*.

- SpatialNotchStep1.
- SpatialNotchsBandwidth1.
- SpatialNotchsGain1.
- SpatialNotchStep2.
- SpatialNotchsBandwidth2.
- SpatialNotchsGain2.
- SpatialNotchStep3.
- SpatialNotchsBandwidth3.
- SpatialNotchsGain3.

### **Prototype**

```
int PositionerCompensationSpatialPeriodicNotchsSet(
    int SocketID,
    char * FullPositionerName,
    double SpatialNotchStep1,
    double SpatialNotchBandwidth1,
    double SpatialNotchGain1,
    double SpatialNotchStep2,
    double SpatialNotchBandwidth2,
    double SpatialNotchGain2,
    double SpatialNotchStep3,
    double SpatialNotchBandwidth3,
    double SpatialNotchGain3
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
SpatialNotchStep1	double	Spatial periodic step for filter #1 (units).
SpatialNotchBandwidth1	double	Spatial periodic bandwidth for filter #1 (Hz).
SpatialNotchGain1	double	Spatial periodic gain for filter #1.
SpatialNotchStep2	double	Spatial periodic step for filter #2 (units).
SpatialNotchBandwidth2	double	Spatial periodic bandwidth for filter #2 (Hz).
SpatialNotchGain2	double	Spatial periodic gain for filter #2.
SpatialNotchStep3	double	Spatial periodic step for filter #3 (units).
SpatialNotchBandwidth3	double	Spatial periodic bandwidth for filter #3 (Hz).
SpatialNotchGain3	double	Spatial periodic gain for filter #3.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### 6.2.1.199 PositionerCorrectorAutoTuning

#### Name

**PositionerCorrectorAutoTuning** – Executes auto-tuning process for determining position control loop PID values.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type: (-8)
- Positioner must not be a “Secondary Positioner”: (-8)
- Check positioner name: (-18)
- Group status must be “READY”: (-22)
- Control loop type must be “PIDFFVelocity”, “PIDDualFFVoltage” or “PIDFFAcceleration”: (-24)

#### Description

The function executes an auto-tuning process and returns the calculated PID settings (KP, KI and KD values). The selected group must be in “READY” state, else (-22) error is returned.

This function works only if the positioner control loop type is “PIDFFVelocity” (velocity control), “PIDDualFFVoltage” (voltage control) or “PIDFFAcceleration” (acceleration control), else it returns (-24) error.

If the function is called when the positioner is not in READY state, (-22) error will be returned.

The “Mode” input value indicates the control mode of the position loop (**Short Settle** or **High Robustness**).

In the **Short Settle** mode, the PID values are adjusted to have high motion performance (short settling time, less following errors).

The **High Robustness** mode is used for a relatively good performance in motion, but guarantees the robustness (stability) for all stage situations (positions, velocities, accelerations).

If auto-tuning initialization fails (-104) error is returned, or if the motion becomes disabled then (-26) error is returned.

The auto-tuning process is executed in 5 periods. At the end of each period, the auto-tuning process estimates the auto-tuning quality by calculating the noise/signal ratio. If the noise/signal ratio is very close to zero (it means no oscillation), an (-101) error is returned. Else if the noise ratio >MaximumNoiseRatio (normally between 0.1 and 0.2, exact value defined in system.ref) then (-102) error is returned.

If the number of acquired data points (minimum = 9) or the number of acquired signal periods (minimum = 5) is not enough for a good estimate then (-103) error is returned.

At end of this function, the new PID setting is returned and the group status becomes “READY” once again.

**Prototype**

```
int PositionerCorrectorAutoTuning(
    int SocketID,
    char * PositionerName,
    int Mode,
    double * KP,
    double * KI,
    double * KD
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
Mode	int	Loop control mode (0 = short settle, or 1 = robust).

**Output parameters**

KP	double *	Calculated KP value.
KI	double *	Calculated KI value.
KD	double *	Calculated KD value.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -5: Not allowed due to a positioner error or hardware status.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
- -101: Relay Feedback Test failed: No oscillation.
- -102: Relay Feedback Test failed: Signal too noisy.
- -103: Relay Feedback Test failed: Signal data not enough for analyse.
- -104: Error of tuning process initialization.

### **6.2.1.200 PositionerCorrectorDamperFilterGet**

#### **Name**

**PositionerCorrectorDamperFilterGet** – Returns Dual control loop parameters for a selected positioner.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check damper filter is enabled: (-205)

#### **Description**

This function allows returning the Damper Filter parameters for a PID Acceleration control loop.

#### **NOTE**

**The corrector must be “PIDFFAccelerationCorrector”.**

#### **Prototype**

```
int PositionerCorrectorDamperFilterGet(
    int SocketID,
    char * PositionerName,
    double * CutOffFrequency,
    double * DamperFactor,
    double * Gain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

CutOffFrequency	double *	Damper filter cut off frequency (Hz).
DamperFactor	double *	Damper factor (1 by default).
Gain	double *	Filter gain (0 or negative value).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -205: Not enabled in your configuration.

### **6.2.1.201 PositionerCorrectorDamperFilterSet**

#### **Name**

**PositionerCorrectorDamperFilterSet** – Sets Dual control loop parameters for a selected positioner.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)
- CutOffFrequency  $\geq 0$  and  $\leq (0.5 / ISRCorrectorPeriod)$
- DamperFactor  $> 0$
- Gain  $\leq 0$
- Check the motion status (Motion status must be disable): (-134)
- Check damper filter is enabled: (-205)

#### **Description**

This function allows configuring the Damper Filter parameters for a PID Acceleration control loop.

#### **NOTE**

**The main control loop must be “PIDFFAccelerationCorrector”.**

**This function is enabled only if the group state is DISABLE.**

#### **Prototype**

```
int PositionerCorrectorDamperFilterSet(
    int SocketID,
    char * PositionerName,
    double CutOffFrequency,
    double DamperFactor,
    double Gain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
CutOffFrequency	double	Damper filter cut off frequency (Hz).
DamperFactor	double	Damper factor (1 by default).
Gain	double	Filter gain (0 or negative value).

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -205: Not enabled in your configuration.
- -134: Changing the loop status is allowed in DISABLE state only.

### 6.2.1.202 PositionerCorrectorDualGet

#### Name

**PositionerCorrectorDualGet** – Returns Dual control loop parameters for a selected positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check the positioner name: (-18)
- Check dual corrector is enabled: (-205)

#### Description

This function allows returning the dual control loop parameter values.

---

#### NOTE

The “DualCorrectorMode” must be “Enabled” in the stages.ini file to activate the dual control loop. A secondary encoder has to be configured to be able to use the dual control loop. The main control loop must be “PIDFFAccelerationCorrector”.

---

#### Prototype

```
int PositionerCorrectorDualGet(  
    int SocketID,  
    char * PositionerName,  
    bool * ClosedLoopStatus,  
    double * KP,  
    double * KI,  
    double * KD,  
    double * IntegrationTime,  
    double * DerivativeFilterCutOffFrequency,  
    double * KFeedForwardAcceleration,  
    double * KFeedForwardJerk  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

### **Output parameters**

ClosedLoopStatus	bool *	Dual control loop status (true = closed and false = opened).
KP	double *	PID servo loop proportional gain.
KI	double *	PID servo loop integral gain.
KD	double *	PID servo loop derivative gain.
IntegrationTime	double *	PID integration time (seconds).
DerivativeFilterCutOffFrequency	double *	PID derivative filter cut off frequency (Hz).
KFeedForwardAcceleration	double *	Acceleration feedforward gain (units).
KFeedForwardJerk	double *	Jerk feedforward gain (units).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -205: Not enabled in your configuration.

### 6.2.1.203 PositionerCorrectorDualSet

#### Name

**PositionerCorrectorDualSet** – Sets Dual control loop parameters for a selected positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check input parameters value: (-17)  
 $KP \geq 0, KI \geq 0, KD \geq 0, KFeedForwardAcceleration \geq 0, KFeedForwardJerk \geq 0$   
 $IntegrationTime \geq CorrectorPeriod (0.0001 \text{ s})$
- Check the positioner name: (-18)

#### Description

This function allows returning the dual control loop parameter values.

---

#### NOTE

The “DualCorrectorMode” must be “Enabled” in the stages.ini file to activate the dual control loop. A secondary encoder has to be configured to be able to use the dual control loop. The main control loop must be “PIDFFAccelerationCorrector”.

---

#### Prototype

```
int PositionerCorrectorDualSet(  
    int SocketID,  
    char * PositionerName,  
    bool ClosedLoopStatus,  
    double KP,  
    double KI,  
    double KD,  
    double IntegrationTime,  
    double DerivativeFilterCutOffFrequency,  
    double KFeedForwardAcceleration,  
    double KFeedForwardJerk  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
ClosedLoopStatus	bool	Dual control loop status (true = closed and false = opened).
KP	double	PID servo loop proportional gain.
KI	double	PID servo loop integral gain.
KD	double	PID servo loop derivative gain.
IntegrationTime	double	PID integration time (seconds).
DerivativeFilterCutOffFrequency	double	PID derivative filter cut off frequency (Hz).
KFeedForwardAcceleration	double	Acceleration feedforward gain (units).
KFeedForwardJerk	double	Jerk feedforward gain (units).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### 6.2.1.204 PositionerCorrectorExcitationSignalGainGet

#### Name

**PositionerCorrectorExcitationSignalGainGet** – Returns corrector excitation signal gain.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)

#### Description

This function returns the corrector excitation signal gain for the selected positioner.

#### Prototype

```
int PositionerCorrectorPIDBaseGet(  
    int SocketID,  
    char * PositionerName,  
    double * ExcitationSignalGain  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

ExcitationSignalGain	double *	Excitation signal gain.
----------------------	----------	-------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.205 PositionerCorrectorExcitationSignalGainSet**

#### **Name**

**PositionerCorrectorExcitationSignalGainSet** – Sets corrector excitation signal gain.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the parameter value: value must be between -1 and 1, else return “Parameter out of range or incorrect” (-17)
- Check the positioner name: (-18)

#### **Description**

This function configures the corrector excitation signal gain for the selected positioner. ExcitationSignalGain value must  $\geq -1$  and  $\leq 1$ .

#### **NOTE**

**The final gain of excitation signal for primary positioner = KFeedforwardExcitationToPrimary \* ExcitationSignalGain**

**The final gain of excitation signal for secondary positioner = KFeedforwardExcitationToSecondary \* ExcitationSignalGain**

**To have more details about KFeedforwardExcitationToPrimary and KFeedforwardExcitationToSecondary , refer to the API PositionerExcitationKFeedforwardSet()**

#### **Prototype**

**Int PositionerCorrectorExcitationSignalGainSet(**

```
    int SocketID,
    char * PositionerName,
    double ExcitationSignalGain
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
ExcitationSignalGain	double	Excitation signal gain.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### 6.2.1.206 PositionerCorrectorNotchFiltersGet

#### Name

**PositionerCorrectorNotchFiltersGet** – Returns the notch filter parameters.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the positioner type: (-8)

#### Description

This function returns the parameters defined for two notch filters.

First notch filter parameters:

- UserNotchFrequency1.
- UserNotchBandwidth1.
- UserNotchGain1.

Second notch filter parameters:

- UserNotchFrequency2.
- UserNotchBandwidth2.
- UserNotchGain2.

---

#### NOTE

If the corrector type is "NoEncoderPositionCorrector" then (-24) error is returned.

---

#### Prototype

```
int PositionerCorrectorNotchFiltersGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * NotchFrequency1,  
    double * NotchBandwidth1,  
    double * NotchGain1,  
    double * NotchFrequency2,  
    double * NotchBandwidth2,  
    double * NotchGain2  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
FullPositionerName	char *	Positioner name.

### **Output parameters**

NotchFrequency1	double * Frequency (Hertz) for notch filter #1.
NotchBandwidth1	double * Band width (Hertz) for notch filter #1.
NotchGain1	double * Gain for notch filter #1.
NotchFrequency2	double * Frequency (Hertz) for notch filter #2.
NotchBandwidth2	double * Band width (Hertz) for notch filter #2.
NotchGain2	double * Gain for notch filter #2.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.207 PositionerCorrectorNotchFiltersSet

#### Name

**PositionerCorrectorNotchFiltersSet** – Sets the notch filter parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check parameter values: (-17)

$$\bullet \text{ NotchFrequency } \in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$$

$$\bullet \text{ NotchBandwidth } \in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$$

$$\bullet \text{ NotchGain } \in [0:100]$$

---

#### NOTE

Refer to *system.ref* file to get **CorrectorISRPeriod** value.

---

#### Description

This function configures the parameters defined for two notch filters. If the “NotchFrequency” value is NULL or the “NotchGain” value is NULL then the notch filter is not activated.

First notch filter parameters:

- NotchFrequency1.
- NotchBandwidth1.
- NotchGain1.

Second notch filter parameters:

- NotchFrequency2.
- NotchBandwidth2.
- NotchGain2.

---

#### NOTE

If the corrector type is “**NoEncoderPositionCorrector**” then (-24) error is returned.

---

### **Prototype**

```
int PositionerCorrectorNotchFiltersSet(
    int SocketID,
    char * FullPositionerName,
    double NotchFrequency1,
    double NotchBandwith1,
    double NotchGain1,
    double NotchFrequency2,
    double NotchBandwith2,
    double NotchGain2
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
NotchFrequency1	double	Frequency (Hertz) for notch filter #1.
NotchBandwith1	double	Band width (Hertz) for notch filter #1.
NotchGain1	double	Gain for notch filter #1.
NotchFrequency2	double	Frequency (Hertz) for notch filter #2.
NotchBandwith2	double	Band width (Hertz) for notch filter #2.
NotchGain2	double	Gain for notch filter #2.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.208 PositionerCorrectorPIDAccelerationFilterGet

#### Name

**PositionerCorrectorPIDAccelerationFilterGet** – Returns PID acceleration corrector filter parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)

#### Description

This function returns PID acceleration corrector filter parameters.

#### Prototype

```
int PositionerCorrectorPIDAccelerationFilterGet(  
    int SocketID,  
    char * PositionerName,  
    bool * FilterControlStatus,  
    double * KD,  
    double * DerivativeFilterCutOffFrequency  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

FilterControlStatus	bool *	Current position of X axis.
KD	double *	KD gain.
DerivativeFilterCutOffFrequency	double *	Derivative Filter Cut Off Frequency (Hz).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.209 PositionerCorrectorPIDAccelerationFilterSet**

#### **Name**

**PositionerCorrectorPIDAccelerationFilterSet** – Sets PID acceleration corrector filter parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check the motion status (Motion status must be disable): (-134)
- Check PID Acceleration Filter is enabled: (-205)

#### **Description**

This function updates PID acceleration corrector filter parameters and enables/disables the filter.

#### **Prototype**

```
int PositionerCorrectorPIDAccelerationFilterSet(
    int SocketID,
    char * PositionerName,
    bool * FilterControlStatus,
    double * KD,
    double * DerivativeFilterCutOffFrequency
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
FilterControlStatus	bool	Current position of X axis.
KD	double	KD gain.
DerivativeFilterCutOffFrequency	double	Derivative Filter Cut Off Frequency (Hz).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -134: Changing the loop status is allowed in DISABLE state only.
- -205: Not enabled in your configuration.

### **6.2.1.210 PositionerCorrectorPIDBaseGet**

#### **Name**

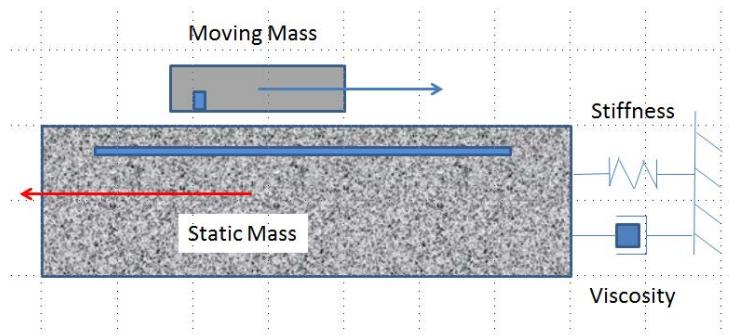
**PositionerCorrectorPIDBaseGet** – Returns PIDBase parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)

#### **Description**

This function returns the PIDBase parameter values.



PIDBase parameters:

- MovingMass.
- StaticMass.
- Viscosity.
- Stiffness.

#### **Prototype**

```
int PositionerCorrectorPIDBaseGet(
    int SocketID,
    char * PositionerName,
    double * MovingMass,
    double * StaticMass,
    double * Viscosity,
    double * Stiffness
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

MovingMass	double *	Mass of the stage moving part.
StaticMass	double *	Mass of the stage static part (the base).
Viscosity	double *	Viscosity of the base.
Stiffness	double *	Stiffness of the base.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.

### 6.2.1.211 PositionerCorrectorPIDBaseSet

#### Name

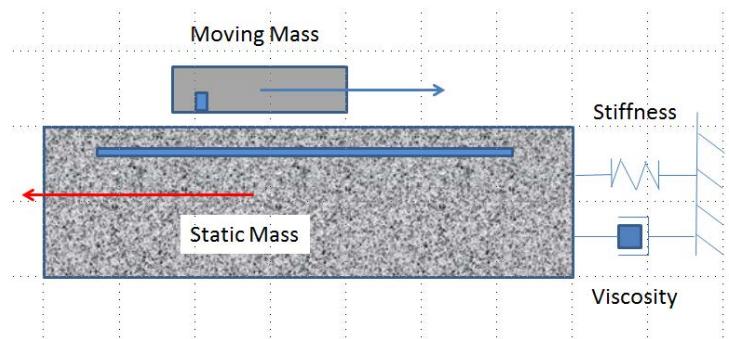
**PositionerCorrectorPIDBaseSet** – Sets PIDBase parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the parameter values: all values must  $\geq 0$ , else return “Parameter out of range or incorrect” (-17)
- Check the positioner name: (-18)

#### Description

This function configures the PIDBase parameters defined for the selected positioner.



PIDBase parameters to set:

- MovingMass.
- StaticMass.
- Viscosity.
- Stiffness.

#### Prototype

```
int PositionerCorrectorPIDBaseSet(
    int SocketID,
    char * PositionerName,
    double MovingMass,
    double StaticMass,
    double Viscosity,
    double Stiffness
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
MovingMass	double	Mass of the stage moving part.
StaticMass	double	Mass of the stage static part (the base).
Viscosity	double	Viscosity of the base.
Stiffness	double	Stiffness of the base.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.

### 6.2.1.212 PositionerCorrectorPIDDualFFVoltageGet

#### Name

**PositionerCorrectorPIDDualFFVoltageGet** – Gets the corrector “PIDDualFFVoltage” parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type and the corrector type: (-8)

#### Description

This function returns the corrector parameter values used by a PID dual feed-forward with a motor voltage output.

---

#### NOTE

The “CorrectorType” must be “PIDDualFFVoltage” in the stages.ini file. This servo loop type is used when the position servo loop drives the voltage applied directly to the motor.

---

#### Prototype

```
int PositionerCorrectorPIDDualFFVoltageGet(  
    int SocketID,  
    char * FullPositionerName,  
    bool * ClosedLoopStatus,  
    double * KP,  
    double * KI,  
    double * KD,  
    double * KS,  
    double * IntegrationTime,  
    double * DerivativeFilterCutOffFrequency,  
    double * GKP,  
    double * GKI,  
    double * GKD,  
    double * KForm,  
    double * FeedForwardGainVelocity,  
    double * FeedForwardGainAcceleration,  
    double * Friction  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

### **Output parameters**

ClosedLoopStatus	bool *	Position servo loop status (true = closed and false = opened).
KP	double *	PID servo loop proportional gain.
KI	double *	PID servo loop integral gain.
KD	double *	PID servo loop derivative gain.
KS	double *	PID integral saturation value (0 to 1).
IntegrationTime	double *	PID integration time (seconds).
DerivativeFilterCutOffFrequency	double *	PID derivative filter cut off frequency (Hz).
GKP	double *	Variable PID proportional gain multiplier.
GKI	double *	Variable PID integral gain multiplier.
GKD	double *	Variable PID derivative gain multiplier.
KForm	double *	Variable PID form coefficient.
FeedForwardGainVelocity	double *	Velocity feedforward gain (units).
FeedForwardGainAcceleration	double *	Acceleration feedforward gain (units).
Friction	double *	Friction compensation.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.213 PositionerCorrectorPIDDualFFVoltageSet**

#### **Name**

**PositionerCorrectorPIDDualFFVoltageSet** – Configures the corrector “PIDDualFFVoltage” parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type and the corrector type: (-8)
- Check parameter value: (-17)
  - $K_P \geq 0$ .
  - $K_I \geq 0$ .
  - $K_D \geq 0$ .
  - $0 \leq K_S \leq 1$ .
  - $\text{IntegrationTime} \geq \text{CorrectorISRPeriod}$ .
  - $G_K_P > -1$ .
  - $G_K_I > -1$ .
  - $G_K_D > -1$ .
  - $K_{Form} \geq 0$ .
  - $K_{FeedForwardVelocity} \geq 0$ .
  - $K_{FeedForwardAcceleration} \geq 0$ .
  - $Friction \geq 0$ .
  - $\text{DerivativeFilterCutOffFrequency} \in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$

---

#### **NOTE**

Refer to *system.ref* file to get **CorrectorISRPeriod** value.

---

#### **Description**

This function configures the “PIDDualFFVoltage” corrector parameters. The “CorrectorType” must be “PIDDualFFVoltage” in the stages.ini file, else (-8) error is returned.

---

#### **NOTE**

**This servo loop type is used when the position servo loop drives the voltage applied directly to the motor.**

---

**Prototype**

```
int PositionerCorrectorPIIDualFFVoltageSet(
    int SocketID,
    char * FullPositionerName,
    bool ClosedLoopStatus,
    double KP,
    double KI,
    double KD,
    double KS,
    double IntegrationTime,
    double DerivativeFilterCutOffFrequency,
    double GKP,
    double GKI,
    double GKD,
    double KForm,
    double FeedForwardGainVelocity,
    double FeedForwardGainAcceleration,
    double Friction
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” unction.
FullPositionerName	char *	Positioner name.
ClosedLoopStatus	bool	Position servo loop status (true = closed and false = opened).
KP	double	PID servo loop proportional gain.
KI	double	PID servo loop integral gain.
KD	double	PID servo loop derivative gain.
KS	double	PID integral saturation value (0 to 1).
IntegrationTime	double	PID integration time (seconds).
DerivativeFilterCutOffFrequency	double	PID derivative filter cut off frequency (Hz).
GKP	double	Variable PID proportional gain multiplier.
GKI	double	Variable PID integral gain multiplier.
GKD	double	Variable PID derivative gain multiplier.
KForm	double	Variable PID form coefficient.
FeedForwardGainVelocity	double	Velocity feedforward gain (units).
FeedForwardGainAcceleration	double	Acceleration feedforward gain (units).
Friction	double	Friction compensation.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.214 PositionerCorrectorPIDFFAccelerationGet**

#### **Name**

**PositionerCorrectorPIDFFAccelerationGet** – Gets the corrector “PIDFFAcceleration” parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type and the corrector type: (-8)

#### **Description**

This function returns the corrector parameter values used by a PID feed-forward with an acceleration output.

#### **NOTE**

The “CorrectorType” must be “PIDFFAcceleration” in the stages.ini file. This servo loop type is used when a constant value applied to the driver results in a constant acceleration of the stage.

#### **Prototype**

```
int PositionerCorrectorPIDFFAccelerationGet(
    int SocketID,
    char * FullPositionerName,
    bool * ClosedLoopStatus,
    double * KP,
    double * KI,
    double * KD,
    double * KS,
    double * IntegrationTime,
    double * DerivativeFilterCutOffFrequency,
    double * GKP,
    double * GKI,
    double * GKD,
    double * KForm,
    double * FeedForwardGainAcceleration,
    double * FeedForwardGainJerk
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

### **Output parameters**

ClosedLoopStatus	bool *	Position servo loop status (true = closed and false = opened).
KP	double *	PID servo loop proportional gain.
KI	double *	PID servo loop integral gain.
KD	double *	PID servo loop derivative gain.
KS	double *	PID integral saturation value (0 to 1).
IntegrationTime	double *	PID integration time (seconds).
DerivativeFilterCutOffFrequency	double *	PID derivative filter cut off frequency (Hz).
GKP	double *	Variable PID proportional gain multiplier.
GKI	double *	Variable PID integral gain multiplier.
GKD	double *	Variable PID derivative gain multiplier.
KForm	double *	Variable PID form coefficient.
FeedForwardGainAcceleration	double *	Velocity feedforward gain (units).
FeedForwardGainJerk	double *	Acceleration feedforward gain (units).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.215 PositionerCorrectorPIDFFAccelerationSet**

#### **Name**

**PositionerCorrectorPIDFFAccelerationSet** – Sets the corrector “PIDFFAcceleration” parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type and the corrector type: (-8)
- Check parameter value: (-17)
  - $KP \geq 0$ .
  - $KI \geq 0$ .
  - $KD \geq 0$ .
  - $0 \leq KS \leq 1$ .
  - $IntegrationTime \geq CorrectorISRPeriod$ .
  - $GKP > -1$ .
  - $GKI > -1$ .
  - $GKD > -1$ .
  - $KForm \geq 0$ .
  - $KFeedForwardAcceleration \geq 0$ .
  - $KFeedForwardJerk \geq 0$ .
  - $DerivativeFilterCutOffFrequency \in \left[ 0 : \frac{0.5}{CorrectorISRPeriod} \right]$

---

#### **NOTE**

Refer to *system.ref* file to get **CorrectorISRPeriod** value.

---

#### **Description**

This function configures the “PIDFFAcceleration” corrector parameters.

---

#### **NOTE**

The “**CorrectorType**” parameter must be defined as “**PIDFFAcceleration**” in the “**stages.ini**” file else (-8) error is returned. This servo loop type is used when a constant value applied to the driver results in a constant acceleration of the stage.

---

**Prototype**

```
int PositionerCorrectorPIDFFAccelerationSet(
    int SocketID,
    char * FullPositionerName,
    bool ClosedLoopStatus,
    double KP,
    double KI,
    double KD,
    double KS,
    double IntegrationTime,
    double DerivativeFilterCutOffFrequency,
    double GKP,
    double GKI,
    double GKD,
    double KForm,
    double FeedForwardGainAcceleration,
    double FeedForwardGainJerk
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
ClosedLoopStatus	bool	Position servo loop status (true = closed and false = opened).
KP	double	PID servo loop proportional gain.
KI	double	PID servo loop integral gain.
KD	double	PID servo loop derivative gain.
KS	double	PID integral saturation value (0 to 1).
IntegrationTime	double	PID integration time (seconds).
DerivativeFilterCutOffFrequency	double	PID derivative filter cut off frequency (Hz).
GKP	double	Variable PID proportional gain multiplier.
GKI	double	Variable PID integral gain multiplier.
GKD	double	Variable PID derivative gain multiplier.
KForm	double	Variable PID form coefficient.
FeedForwardGainAcceleration	double	Velocity feedforward gain (units).
FeedForwardGainJerk	double	Acceleration feedforward gain (units).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.216 PositionerCorrectorPIDFFVelocityGet**

#### **Name**

**PositionerCorrectorPIDFFVelocityGet** – Gets the corrector “PIDFFVelocity” parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type and the corrector type: (-8)

#### **Description**

This function returns the corrector parameter values used by a PID with a velocity output:

ClosedLoopStatus, KP, KI, KD, KS, IntegrationTime, DerivativeFilterCutOffFrequency, GKP, GKI, GKD, Kform and FeedForwardGainVelocity.

#### **NOTE**

**The “CorrectorType” must be “PIDFFVelocity” in the stages.ini file. This servo loop type is used when a constant value applied to the driver results in a constant velocity of the stage.**

#### **Prototype**

```
int PositionerCorrectorPIDFFVelocityGet(
    int SocketID,
    char * FullPositionerName,
    bool * ClosedLoopStatus,
    double * KP,
    double * KI,
    double * KD,
    double * KS,
    double * IntegrationTime,
    double * DerivativeFilterCutOffFrequency,
    double * GKP,
    double * GKI,
    double * GKD,
    double * KForm,
    double * FeedForwardGainVelocity
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer”function.
FullPositionerName	char *	Positioner name.

### **Output parameters**

ClosedLoopStatus	bool *	Position servo loop status (true = closed and false = opened).
KP	double *	PID servo loop proportional gain.
KI	double *	PID servo loop integral gain.
KD	double *	PID servo loop derivative gain.
KS	double *	PID integral saturation value (0 to 1).
IntegrationTime	double *	PID integration time (seconds).
DerivativeFilterCutOffFrequency	double *	PID derivative filter cut off frequency (Hz).
GKP	double *	Variable PID proportional gain multiplier.
GKI	double *	Variable PID integral gain multiplier.
GKD	double *	Variable PID derivative gain multiplier.
KForm	double *	Variable PID form coefficient.
FeedForwardGainVelocity	double *	Velocity feedforward gain (units).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.217 PositionerCorrectorPIDFFVelocitySet

#### Name

**PositionerCorrectorPIDFFVelocitySet** – Configures the corrector “PIDFFVelocity” parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type and the corrector type: (-8)
- Check parameter value: (-17)
  - $KP \geq 0$ .
  - $KI \geq 0$ .
  - $KD \geq 0$ .
  - $0 \leq KS \leq 1$ .
  - $IntegrationTime \geq CorrectorISRPeriod$ .
  - $GKP > -1$ .
  - $GKI > -1$ .
  - $GKD > -1$ .
  - $KForm \geq 0$ .
  - $KFeedForwardVelocity \geq 0$ .
  - $DerivativeFilterCutOffFrequency \in \left[ 0 : \frac{0.5}{CorrectorISRPeriod} \right]$

---

#### NOTE

Refer to *system.ref* file to get **CorrectorISRPeriod** value.

---

#### Description

This function configures the “PIDFFVelocity” corrector parameters.

---

#### NOTE

The “CorrectorType” parameter must be defined as “PIDFFVelocity” in the stages.ini file else (-8) error is returned. This servo loop type is used when a constant value applied to the driver results in a constant velocity of the stage.

---

**Prototype**

```
int PositionerCorrectorPIDFFVelocitySet(
    int SocketID,
    char * FullPositionerName,
    bool ClosedLoopStatus,
    double KP,
    double KI,
    double KD,
    double KS,
    double IntegrationTime,
    double DerivativeFilterCutOffFrequency,
    double GKP,
    double GKI,
    double GKD,
    double KForm,
    double FeedForwardGainVelocity
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
ClosedLoopStatus	bool	Position servo loop status (true = closed and false = opened).
KP	double	PID servo loop proportional gain.
KI	double	PID servo loop integral gain.
KD	double	PID servo loop derivative gain.
KS	double	PID integral saturation value (0 to 1).
IntegrationTime	double	PID integration time (seconds).
DerivativeFilterCutOffFrequency	double	PID derivative filter cut off frequency (Hz).
GKP	double	Variable PID proportional gain multiplier.
GKI	double	Variable PID integral gain multiplier.
GKD	double	Variable PID derivative gain multiplier.
KForm	double	Variable PID form coefficient.
FeedForwardGainVelocity	double	Velocity feedforward gain (units).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.218 PositionerCorrectorPIPositionGet**

#### **Name**

**PositionerCorrectorPIPositionGet** – Returns the corrector “PIPosition” parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type and the corrector type: (-8)

#### **Description**

This function returns the corrector parameter values used by a PI with a position output:  
ClosedLoopStatus, KP, KI and IntegrationTime.

#### **NOTE**

**The “CorrectorType” must be “PIPosition” in the stages.ini file. This servo loop type is used when the position servo loop outputs a position value directly.**

#### **Prototype**

```
int PositionerCorrectorPIPositionGet(
    int SocketID,
    char * FullPositionerName,
    bool * ClosedLoopStatus,
    double * KP,
    double * KI,
    double * IntegrationTime
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

ClosedLoopStatus	bool *	Position servo loop status (true = closed and false = opened).
KP	double *	PID servo loop proportional gain.
KI	double *	PID servo loop integral gain.
KD	double *	PID servo loop derivative gain.
IntegrationTime	double *	PID integration time (seconds).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.219 PositionerCorrectorPIPositionSet

#### Name

**PositionerCorrectorPIPositionSet** – Configures the corrector “PIPosition” parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type and the corrector type: (-8)
- Check parameter value: (-17)
- KP  $\geq 0$ .
- KI  $\geq 0$ .
- IntegrationTime  $\geq$  CorrectorISRPeriod.

---

#### NOTE

Refer to *system.ref* file to get CorrectorISRPeriod value.

---

#### Description

This function configures the “PIPosition” corrector parameters.

---

#### NOTE

The “CorrectorType” parameter must be defined as “PIPosition” in the stages.ini file else ERR\_WRONG\_OBJECT\_TYPE (-8) is returned. This servo loop type is used when the position servo loop outputs a position value directly.

---

#### Prototype

```
int PositionerCorrectorPIPositionSet(
    int SocketID,
    char * FullPositionerName,
    bool ClosedLoopStatus,
    double KP,
    double KI,
    double IntegrationTime
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
ClosedLoopStatus	bool	Position servo loop status (true = closed and false = opened).
KP	double	PID servo loop proportional gain.
KI	double	PID servo loop integral gain.
KD	double	PID servo loop derivative gain.
IntegrationTime	double	PID integration time (seconds).

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.220 PositionerCorrectorPlantFeedForwardDelayGet**

#### **Name**

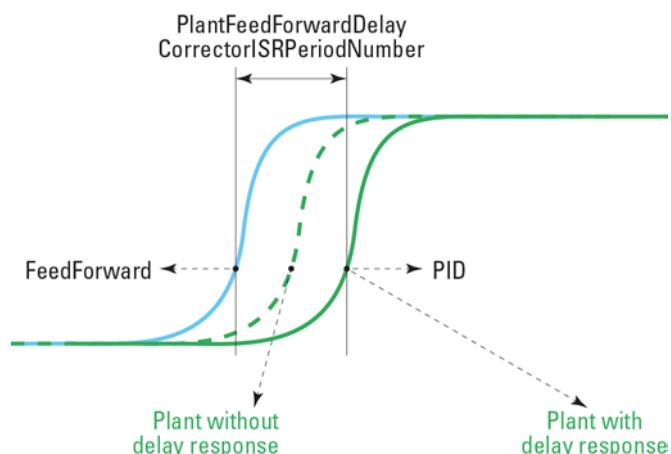
**PositionerCorrectorPlantFeedForwardDelayGet** – Returns the corrector ISR period number configured to define the plant delay.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the filter number: (-17)

#### **Description**

This function returns the corrector ISR period number configured to define the plant delay. Its value is predefined in the stages.ini file with the parameter named “PlantFeedForwardDelayCorrectorISRPeriodNumber”.



#### **Prototype**

```
int PositionerCorrectorPlantFeedForwardDelayGet(
    int SocketID,
    char * PositionerName,
    int * CorrectorISRPeriodNumber
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

CorrectorISRPeriodNumber int \* Number of ISR periods to delay plant

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.221 PositionerCorrectorPlantFeedForwardDelaySet**

#### **Name**

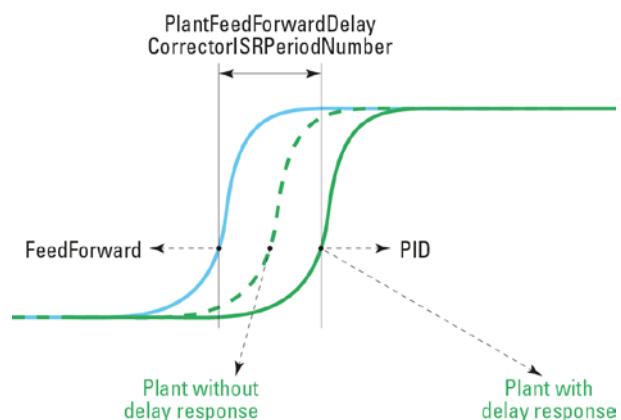
**PositionerCorrectorPlantFeedForwardDelaySet** – Sets the corrector ISR period number that defines the plant delay.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the filter number: (-17)

#### **Description**

This function reconfigures the corrector ISR period number to define the plant delay. Its value is predefined in stages.ini with the parameter “PlantFeedForwardDelayCorrectorISRPeriodNumber”.



#### **Prototype**

```
int PositionerCorrectorPlantFeedForwardDelaySet(
    int SocketID,
    char * PositionerName,
    int CorrectorISRPeriodNumber )
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
CorrectorISRPeriodNumber	int	Number of ISR periods to delay plant (max = 100).

#### **Output parameters**

None.

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### 6.2.1.222 PositionerCorrectorPostFFGet

#### Name

**PositionerCorrectorPostFFGet** – Returns parameters of Post Feed Forward from PIDFFAccelection corrector.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name: (-18)

#### Description

This function returns the parameters of the current post feed forward from PIDFFAccelection corrector.

#### Prototype

```
int PositionerCorrectorPostFFGet(  
    int SocketID,  
    char * PositionerName,  
    double * PostKFeedForwardAcceleration,  
    double * PostKFeedForwardJerk,  
    double * PostKFeedForwardSlope  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

PostKFeedForwardAcceleration	double *	Post KFeedForward acceleration
PostKFeedForwardJerk	double *	Post KFeedForward jerk
PostKFeedForwardSlope	double *	Post KFeedForward slop

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -18: Positioner name doesn't exist or incorrect.

### **6.2.1.223 PositionerCorrectorPostFFSet**

#### **Name**

**PositionerCorrectorPostFFSet** – Sets parameters of Post Feed Forward from PIDFFAccelection corrector.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name: (-18)
- Not allowed due to configuration disabled: (-121)

#### **Description**

This function sets the parameters of the Post Feed Forward from PIDFFAccelection corrector.

#### **Prototype**

```
int PositionerCorrectorPostFFSet(
    int SocketID,
    char * PositionerName,
    double PostKFeedForwardAcceleration,
    double PostKFeedForwardJerk,
    double PostKFeedForwardSlope
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
PostKFeedForwardAcceleration	double	Post KFeedForward acceleration
PostKFeedForwardJerk	double	Post KFeedForward jerk
PostKFeedForwardSlope	double	Post KFeedForward slop

#### **Output parameters**

None.

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -18: Positioner name doesn't exist or incorrect.
- -121: Function is not allowed due to configuration disabled.

### 6.2.1.224 PositionerCorrectorTypeGet

#### Name

**PositionerCorrectorTypeGet** – Returns the corrector type.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)

#### Description

This function returns the corrector type used by the selected positioner.

The corrector type can be one of this list:

- PositionerCorrectorPIDFFAcceleration.
- PositionerCorrectorPIDFFVelocity.
- PositionerCorrectorPIDDualFFVoltage.
- PositionerCorrectorPIPosition.
- NoCorrector.

---

#### NOTE

**The corrector type is defined in the stages.ini file with the “CorrectorType” parameter.**

---

#### Prototype

```
int PositionerCorrectorTypeGet(  
    int SocketID,  
    char * FullPositionerName,  
    char * CorrectorType  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

CorrectorType	char *	Corrector type.
---------------	--------	-----------------

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.225 PositionerCurrentVelocityAccelerationFiltersGet**

#### **Name**

**PositionerCurrentVelocityAccelerationFiltersGet** – Returns the velocity and acceleration filter cut off frequencies.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)

#### **Description**

This function returns the current velocity cut off frequency and the current acceleration cut off frequency used by gathering for the selected positioner.

Gathering uses these parameters to filter the current velocity and the current acceleration. These parameters are defined in the stages.ini file.

#### **Prototype**

```
int PositionerCurrentVelocityAccelerationFiltersGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * VelocityCutOffFrequency,  
    double * AccelerationCutOffFrequency  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

VelocityCutOffFrequency	double *	Velocity filter cut off frequency (Hz).
AccelerationCutOffFrequency	double *	Acceleration filter cut off frequency (Hz).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.226 PositionerCurrentVelocityAccelerationFiltersSet**

#### **Name**

**PositionerCurrentVelocityAccelerationFiltersSet** – Sets the velocity and acceleration filter cut off frequencies.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check parameter value: (-17)

- VelocityCutOffFrequency  $\in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$
- AccelerationCutOffFrequency  $\in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]$

#### **NOTE**

Refer to *system.ref* file to get **CorrectorISRPeriod** value.

#### **Description**

This function sets a new velocity cut off frequency and a new acceleration cut off frequency for the selected positioner.

Gathering uses these parameters to filter the current velocity and the current acceleration. These parameters are defined in the stages.ini file.

#### **Prototype**

```
int PositionerCurrentVelocityAccelerationFiltersSet(
    int SocketID,
    char * FullPositionerName,
    double VelocityCutOffFrequency,
    double AccelerationCutOffFrequency
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
VelocityCutOffFrequency	double	Velocity filter cut off frequency (Hz).
AccelerationCutOffFrequency	double	Acceleration filter cut off frequency (Hz).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.227 PositionerDriverFiltersGet**

#### **Name**

**PositionerDriverFiltersGet** – Gets the piezo driver notch and lowpass filters parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check driver type, if not PIEZO: (-24)
- If piezo driver, check if driver is not initialized: (-118)

#### **Description**

This function returns current values of the piezo driver filters parameters (KI, notch frequency, notch bandwidth, notch gain, lowpass frequency).

#### **Prototype**

```
int PositionerDriverFiltersGet(
    int SocketID,
    char * FullPositionerName,
    double * KI,
    double * NotchFrequency,
    double * NotchBandwidth,
    double * NotchGain,
    double * LowpassFrequency
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

KI	double *	Driver KI.
NotchFrequency	double *	Driver notch frequency (Hz).
NotchBandwidth	double *	Driver notch bandwidth (Hz).
NotchGain	double *	Driver notch gain.
LowpassFrequency	double *	Driver lowpass frequency (Hz).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -24: Not available in this configuration (check hardware or configuration).
- -118: Not allowed action driver not initialized.

### 6.2.1.228 **PositionerDriverFiltersSet**

#### Name

**PositionerDriverFiltersSet** – Sets the piezo driver filters parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check parameter value: (-17)
  - $KI \geq 0$ .
  - $NotchFrequency \in \left[0 : \frac{0.5}{CorrectorISRPeriod}\right]$
  - $NotchBandwidth \in \left[0 : \frac{0.5}{CorrectorISRPeriod}\right]$
  - $NotchGain \in [0 : 100]$
  - $LowpassFrequency \in \left[0 : \frac{0.5}{CorrectorISRPeriod}\right]$

#### **NOTE**

Refer to *system.ref* file to get **CorrectorISRPeriod** value.

- Check driver type, if not PIEZO: (-24)
- If the group state is NOTREF or READY: (-117)
- If the driver is not initialized: (-118)

#### Description

This function sets parameters of the driver (KI integral, notch and lowpass filters) for a piezo driver positioner.

#### Prototype

```
int PositionerDriverFiltersSet(
    int SocketID,
    char * FullPositionerName,
    double KI,
    double NotchFrequency,
    double NotchBandwidth,
    double NotchGain,
    double LowpassFrequency
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
KI	double	Driver KI.
NotchFrequency	double	Driver notch frequency (Hz).
NotchBandwidth	double	Driver notch bandwidth (Hz).
NotchGain	double	Driver notch gain.
LowpassFrequency	double	Driver lowpass frequency (Hz).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -24: Not available in this configuration (check hardware or configuration).
- -50: Motor initialization error. Check InitializationAccelerationLevel, ScalingAcceleration, MaximumJerkTime, EncoderResolution or EncoderScalePitch.
- -117: Function is only allowed in DISABLED state.
- -118: Not allowed action driver not initialized.

### 6.2.1.229 PositionerDriverPositionOffsetsGet

#### Name

**PositionerDriverPositionOffsetsGet** – Gets the current value of piezo driver stage and gage position offsets.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check driver type, if not PIEZO: (-24)
- If the group state is NOTREF or READY: (-117)
- If the driver is not initialized: (-118)

#### Description

This function returns current value of the piezo driver position offset parameters (stage position offset, gage position offset).

#### Prototype

```
int PositionerDriverPositionOffsetsGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * StagePositionOffset,  
    double * GagePositionOffset  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

StagePositionOffset	double *	Driver stage position offset (units).
GagePositionOffset	double *	Driver gage position offset (units).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).
- -117: Function is only allowed in DISABLED state.
- -118: Not allowed action driver not initialized.

### 6.2.1.230 PositionerDriverStatusGet

#### Name

**PositionerDriverStatusGet** – Gets the positioner driver status code.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must be not a secondary positioner): (-18), (-24)

#### Description

This function returns the positioner driver status from the driver board.

Use the “PositionerDriverStatusStringGet” function to get the driver status description.

---

#### NOTE

See section 7.7: “Positioner Driver Status List”.

---

#### Prototype

```
int PositionerDriverStatusGet(  
    int SocketID,  
    char * FullPositionerName,  
    unsigned long * PositionerDriverStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

PositionerDriverStatus	unsigned long *	Driver status code.
------------------------	-----------------	---------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.231 PositionerDriverStatusStringGet

#### Name

**PositionerDriverStatusStringGet** – Gets the positioner driver status description.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns a driver status description from a positioner driver status code.

---

#### NOTE

See section 7.7: “Positioner Driver Status List”.

---

#### Prototype

```
int PositionerDriverStatusStringGet(  
    int SocketID,  
    int DriverStatusCode,  
    char * DriverStatusString  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
DriverStatusCode	unsigned long	Driver status code.

#### Output parameters

DriverStatusString	char *	Driver status description.
--------------------	--------	----------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### **6.2.1.232 PositionerEncoderAmplitudeValuesGet**

#### **Name**

**PositionerEncoderAmplitudeValuesGet** – Gets the encoder amplitude values.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner: (-8)
- Check the encoder type (must be “AnalogInterpolated”): (-8)

#### **Description**

This function returns the maximum and current amplitudes values (in volts) of the analog encoder input.



#### **CAUTION**

**The encoder type must be “AnalogInterpolated” in the stages.ini file (“EncoderType” parameter).**

#### **Prototype**

```
int PositionerEncoderAmplitudeValuesGet(
    int SocketID,
    char * FullPositionerName,
    double * MaxSinusAmplitude,
    double * CurrentSinusAmplitude,
    double * MaxCosinusAmplitude,
    double * CurrentCosinusAmplitude
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

MaxSinusAmplitude	double *	Encoder sinus signal maximum amplitude value (Volts).
CurrentSinusAmplitude	double *	Encoder sinus signal current amplitude value (Volts).
MaxCosinusAmplitude	double *	Encoder cosinus signal maximum amplitude value (Volts).
CurrentCosinusAmplitude	double *	Encoder cosinus signal current amplitude value (Volts).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.233 PositionerEncoderCalibrationParametersGet**

#### **Name**

**PositionerEncoderCalibrationParametersGet** – Gets the encoder calibration parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must be not a secondary positioner): (-8)
- Check the encoder type (must be “AnalogInterpolated”): (-8)

#### **Description**

After a calibration of the analog encoder input (by the function “GroupInitializeWithEncoderCalibration”), this function returns the optimum parameters for the analog encoder interface. To take these parameters into account (recommended to achieve best performance), these values must be entered manually in the corresponding section of the stages.ini file. The parameters to set in the stages.ini file are:

- EncoderSinusOffset = ; Volts
- EncoderCosinusOffset = ; Volts
- EncoderDifferentialGain =
- EncoderPhaseCompensation = ; Deg



#### **CAUTION**

The encoder type must be “AnalogInterpolated” in the stages.ini file (“EncoderType” parameter).

#### **Prototype**

```
int PositionerEncoderCalibrationParametersGet(
    int SocketID,
    char * FullPositionerName,
    double * SinusOffset,
    double * CosinusOffset,
    double * DifferentialGain,
    double * PhaseCompensation
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

SinusOffset	double *	Encoder sinus signal offset (Volts).
CosinusOffset	double *	Encoder cosinus signal offset (Volts).
DifferentialGain	double *	Encoder differential gain.
PhaseCompensation	double *	Encoder phase compensation (Deg).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.234 PositionersEncoderIndexDifferenceGet**

#### **Name**

**PositionersEncoderIndexDifferenceGet** – Gets the distance between the two index encoders (gantry).

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a SingleAxis or an XY): (-8)
- Check the positioner type (must not be a secondary positioner): (-8)
- Check the positioner name: (-18)
- Check the positioner type (must be “gantry”): (-24)
- Check the positioner was at least once homed: (-109)

#### **Description**

This function returns the distance between the two encoders indexes of a “primary positioner – secondary positioner” couple. To use this function, the positioner must be configured in “gantry” mode else (-24) error is returned.

For further information about gantry mode, refer to the “SYSTEM – Manual Configuration – Gantry (Secondary Positioners)” section in the XPS user’s manual.

#### **Prototype**

```
int PositionersEncoderIndexDifferenceGet(
    int SocketID,
    char * FullPositionerName
    double * Distance
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

Distance	double *	Distance between the two index encoders (units).
----------	----------	--

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -109: Group need to be homed at least once to use this function (distance measured during home search).

### 6.2.1.235 PositionerErrorGet

#### Name

**PositionerErrorGet** – Returns the positioner error code and clears it.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid positioner name: (-18)
- Valid secondary positioner: (-18)

#### Description

Returns the positioner error code and clears it.

The positioner error codes are listed in section 7.5: “Positioner Error List”. The description of the positioner error code can be obtained with the “GroupPositionerErrorStringGet” function.

---

#### NOTE

**The “PositionerErrorRead” function reads the positioner error without clearing it.**

---

#### Prototype

```
int PositionerErrorGet(  
    int SocketID,  
    char * FullPositionerName,  
    int * PositionerError  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

PositionerError	int *	Positioner error code.
-----------------	-------	------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.236 PositionerErrorRead

#### Name

**PositionerErrorRead** – Returns the positioner error code without clearing it.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid positioner name: (-18)
- Valid secondary positioner: (-18)

#### Description

Returns the positioner error code without clearing it.

The positioner error codes are listed in section 7.5: “Positioner Error List”. The description of the positioner error code can be obtained with the “GroupPositionerErrorStringGet” function.

---

#### NOTE

**The “PositionerErrorGet” function clears the positioner error.**

---

#### Prototype

```
int PositionerErrorRead(  
    int SocketID,  
    char * FullPositionerName,  
    int * PositionerError  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

PositionerError	int *	Positioner error code.
-----------------	-------	------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.237 PositionerErrorStringGet

#### Name

**PositionerErrorStringGet** – Gets the positioner error description.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns a positioner error description from a positioner error code.

---

#### NOTE

See section 7.5: “Positioner Error List”.

---

#### Prototype

```
int PositionerErrorStringGet(  
    int SocketID,  
    char * FullPositionerName,  
    int PositionerErrorCode,  
    char * PositionerErrorString  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
PositionerErrorCode	int	Positioner error code.

#### Output parameters

PositionerErrorString	int *	Positioner error description.
-----------------------	-------	-------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.238 PositionerExcitationSignalGet

#### Name

**PositionerExcitationSignalGet** – Returns the currently used parameters of the excitation signal functionality.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Valid object type: (-8)
- Valid positioner name: (-18)

#### Description

This function gets the last configured excitation signal parameters.

#### Prototype

```
int PositionerExcitationSignalGet(
```

```
    int SocketID,  
    char * FullPositionerName,  
    int * SignalType,  
    double * Frequency,  
    double * Amplitude,  
    double * Time  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

SignalType	int *	Type of signal.
Frequency	double *	Frequency (Hz).
Amplitude	double *	Amplitude (acceleration, velocity or voltage unit).
Time	double *	During time (seconds).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.

### 6.2.1.239 PositionerExcitationSignalSet

#### Name

**PositionerExcitationSignalSet** – Configures and activate the signal of excitation.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Is secondary positioner or has a secondary positioner: (-8)
- Check frequency (must  $\geq 0.1$  and  $\leq 0.5/\text{CorrectorISRPeriod}$ ): (-17)
- Check excitation time (*must  $\geq 4 * \text{CorrectorISRPeriod}$* ): (-17)
- Check signal amplitude [-Acceleration (Velocity or Voltage) limit to Acceleration (Velocity or Voltage) limit]: (-17)
- Check type of signal (0, 1, 2 or 3): (-17)
- Valid positioner name: (-18)
- Check group status (*must be READY*): (-22)
- Valid control loop type: (-24)

#### Description

The excitation signal functionality generates a typical signal (a sine, a blank noise or an echelon signal) that the controller sends to motors to excite the system. In measuring the output signal of the excited system, we can determine some system characteristics, like the system transfer function.

The excitation signal functionality is only available with the stages controlled in acceleration (acceleration control, ex: brushless / linear motors), velocity (velocity control) or in voltage (voltage control). It does not exist with the stages controlled in position (ex: stepper motors).

The excitation-signal function **PositionerExcitationSignalSet** can be executed only when the positioner is in “READY” state. When the excitation-signal process is in progress, the positioner is in the “ExcitationSignal” state. At the end of the process, the positioner returns to “READY” state (see group state diagram).

The **PositionerExcitationSignalSet** function sends an excitation signal to the motor for a brief time. This function is allowed for “PIDFFAcceleration”, “PIDFFVelocity” or “PIDDualFFVoltage” control loops. The parameters to configure are *signal type* (0:sine, 1:echelon,2:random-amplitude,3:random-pulse-width binary-amplitude, integer), *frequency* (Hz, double), *amplitude* (acceleration, velocity or voltage unit, double) and *during time* (seconds, double).

The function effective parameters for each mode are: (here: Limit means AccelerationLimit, VelocityLimit or VoltageLimit)

- Sine signal mode: Frequency ( $\geq 0.1$  and  $\leq 0.5/\text{CorrectorISRPeriod}$ ), Amplitude ( $> 0$  and  $\leq \text{Limit}$ ), Time ( $\geq 4 * \text{CorrectorISRPeriod}$ )
- Echelon signal mode: Amplitude ( $> 0$  and  $\leq \text{Limit}$ , or  $< 0$  and  $\geq -\text{Limit}$ ), Time ( $\geq 4 * \text{CorrectorISRPeriod}$ )
  - + During Time: Signal = Amplitude
  - + End of Time: Signal = 0
- Random-amplitude signal mode: Amplitude ( $> 0$  and  $\leq \text{Limit}$ ), Time ( $> 0$ ), Frequency ( $\geq 0.1$  and  $\leq 0.5/\text{CorrectorISRPeriod}$ )

Signal is generated with a random value at with a period defined by the controller base time (*CorrectorISRPeriod*, default value 0.125 ms), then is filtered with a second order low-pass filter at the cut-off *Frequency* value.

- Random-pulse-width binary-amplitude signal mode:  
*Amplitude* (>0 and  $\leq$ Limit), *Time* ( $\geq$ 4 \* *CorrectorISRPeriod*), *Frequency* ( $\geq$ 0.1 and  $\leq$ 0.5/*CorrectorISRPeriod*).
- Signal is a sequence of pulses (*Signal* = *Amplitude* or = 0) with pulse randomly varied width (multiple of Tbase).
- Frequency* is the controlled system band-width (cut-off frequency), necessary for the PRBS (Pseudo Random Binary Sequence) function configuration.

The function non-effective parameters can accept any value, the value 0 is recommended for simplicity.

#### NOTE

**If during the excitation signal generation the stage position exceeds the user minimum or maximum target positions, the motor excitation command is stopped and an error is returned.**

#### Prototype

```
int PositionerExcitationSignalSet(
    int SocketID,
    char * FullPositionerName,
    int SignalType,
    double Frequency,
    double Amplitude,
    double Time
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
SignalType	int	Type of signal.
Frequency	double	Frequency (Hz).
Amplitude	double	Amplitude (acceleration, velocity or voltage unit).
Time	double	During time (seconds).

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).
- -112: Error of excitation signal generation initialization.

### **6.2.1.240 PositionerFeedforwardAccDisable**

#### **Name**

**PositionerFeedforwardAccDisable** – Disables XY External Feed Forward Acceleration signal.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Valid positioner name: (-18)
- Not allowed due to configuration disabled: (-121)

#### **Description**

Disable XY External Feed Forward Acceleration signal.

This function is reserved for an XY group.

#### **Prototype**

```
int PositionerFeedforwardAccDisable(  
    int SocketID,  
    char * PositionerName  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Not enable in your configuration.

### **6.2.1.241 PositionerFeedforwardAccEnable**

#### **Name**

**PositionerFeedforwardAccEnable** – Enables XY External Feed forward Acceleration signal.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Valid positioner name: (-18)
- Not allowed due to configuration disabled: (-121)
- Check the group is ready: (-135)

#### **Description**

Enable XY External Feed Forward Acceleration signal.

This function is reserved for an XY group.

#### **Prototype**

```
int PositionerFeedforwardAccEnable(  
    int SocketID,  
    char * PositionerName  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Not enable in your configuration.
- -135: Function is not allowed because group is not initialized or not referenced.

### **6.2.1.242 PositionerFeedforwardAccGet**

#### **Name**

**PositionerFeedforwardAccGet** – Gets parameters of XY external feed forward acceleration signal.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Valid positioner name: (-18)
- Not allowed due to configuration disabled: (-121)

#### **Description**

Gets parameters of the current XY external feed forward acceleration signal.

#### **Prototype**

```
int PositionerFeedforwardAccGet(
    int SocketID,
    char * PositionerName,
    char * OutputName1,
    double * Scale1,
    double * Offset1,
    char * OutputName2,
    double * Scale2,
    double * Offset2
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

OutputName1	char *	GPIO Analog Output name #1
Scale1	double *	Signal 1 scale
Offset1	double *	Signal 1 offset
OutputName2	char *	GPIO Analog Output name #2
Scale2	double *	Signal 2 scale
Offset2	double *	Signal 2 offset

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Not enable in your configuration.

### **6.2.1.243 PositionerFeedforwardAccSet**

#### **Name**

**PositionerFeedforwardAccSet** – Sets parameters of XY external feed forward acceleration signal.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Valid positioner name: (-18)
- Not allowed due to configuration disabled: (-121)

#### **Description**

Sets parameters of the current XY external feed forward acceleration signal.

#### **Prototype**

```
int PositionerFeedforwardAccSet(
    int SocketID,
    char * PositionerName,
    char * OutputName1,
    double Scale1,
    double Offset1,
    char OutputName2,
    double Scale2,
    double Offset2
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
OutputName1	char *	GPIO Analog Output name #1
Scale1	double	Signal 1 scale
Offset1	double	Signal 1 offset
OutputName2	char *	GPIO Analog Output name #2
Scale2	double	Signal 2 scale
Offset2	double	Signal 2 offset

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.  
Not enable in your configuration.

-121:

### 6.2.1.244 PositionerFeedforwardPositionDisable

#### Name

**PositionerFeedforwardPositionDisable** – Disables XY External Feed forward Position signal.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Check the positioner name: (-18)
- Not allowed due to configuration disabled: (-121)

#### Description

Disables XY External Feed forward Position signal.

This function is reserved for an XY group.

#### Prototype

```
int PositionerFeedforwardPositionDisable(  
    int SocketID,  
    char * PositionerName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Not enable in your configuration.

### **6.2.1.245 PositionerFeedforwardPositionEnable**

#### **Name**

**PositionerFeedforwardPositionEnable** – Enables XY External Feed Forward Position signal.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Check the positioner name: (-18)
- Not allowed due to configuration disabled: (-121)
- Check the group status is “READY”: (-135)

#### **Description**

Enables XY External Feed Forward Position signal.

This function is reserved for an XY group.

#### **Prototype**

```
int PositionerFeedforwardPositionEnable(  
    int SocketID,  
    char * PositionerName  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Not enable in your configuration.
- -135: Function is not allowed because group is not initialized or not referenced.

### **6.2.1.246 PositionerFeedforwardPositionGet**

#### **Name**

**PositionerFeedforwardPositionGet** – Gets parameters of XY External Feed forward Position signal.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Check the positioner name: (-18)
- Not allowed due to configuration disabled: (-121)

#### **Description**

Gets parameters of XY External Feed forward Position signal.

This function is reserved for an XY group.

#### **Prototype**

```
int PositionerFeedforwardPositionGet(
    int SocketID,
    char * PositionerName,
    char * OutputName,
    double * Scale,
    double * Offset
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

OutputName	char *	GPIO Analog Output name
Scale	double *	Signal scale
Offset	double *	Signal offset

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Not enable in your configuration.

### 6.2.1.247 PositionerFeedforwardPositionSet

#### Name

**PositionerFeedforwardPositionSet** – Sets parameters of XY External Feed forward Position signal.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Check the positioner name: (-18)
- Not allowed due to configuration disabled: (-121)

#### Description

Sets parameters of XY External Feed forward Position signal.

This function is reserved for an XY group.

#### Prototype

```
int PositionerFeedforwardPositionSet(  
    int SocketID,  
    char * PositionerName,  
    char * OutputName,  
    double Scale,  
    double Offset  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
OutputName	char *	GPIO analog output name
Scale	double	Signal scale
Offset	double	Signal offset

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -121: Not enable in your configuration.

### **6.2.1.248 PositionerGantryEndReferencingPositionGet**

#### **Name**

**PositionerGantryEndReferencingPositionGet** – Gets the secondary axis position at the end of group home search.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner type (must be a primary positioner): (-8)
- Check group type: (-8)
  - SingleAxis.
  - SingleAxisTheta.
  - SingleAxisWithClamping.
  - XY.
  - MultipleAxes.
- Check positioner name: (-18)
- Check if positioner has a SecondaryPositioner (must be gantry): (-24)
- Check if homing is done: (-109)

#### **Description**

This function gets the saved corrected Setpoint position of the secondary positioner at the end of group home search.

#### **Prototype**

```
int PositionerGantryEndReferencingPositionGet(
    int SocketID,
    char PositionerName,
    double * Position
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	int	Positioner name.

#### **Output parameters**

Position	double *	SecondaryPositioner Setpoint Position at the end of home (units).
----------	----------	---

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
- -109: Group need to be homed at least once to use this function (distance measured during home search).

### **6.2.1.249 PositionerHardInterpolatorFactorGet**

#### **Name**

**PositionerHardInterpolatorFactorGet** – Gets the interpolation factor for position compare mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must be not a secondary positioner): (-8)
- Check the encoder type (must be “AnalogInterpolated”): (-8)

#### **Description**

This function returns the interpolation factor of the hardware interpolator used in the “Position Compare” mode. The interpolation factor value is defined as:

InterpolationFactor = round (EncoderScalePitch/HardInterpolatorResolution)

---

#### **NOTE**

**The encoder type must be “AnalogInterpolated” in the stages.ini file (“EncoderType” parameter).**

---

#### **Prototype**

```
int PositionerHardInterpolatorFactorGet(  
    int SocketID,  
    char * FullPositionerName,  
    int * InterpolationFactor  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
Output parameters.		
InterpolationFactor	int *	interpolation factor.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.250 PositionerHardInterpolatorFactorSet**

#### **Name**

**PositionerHardInterpolatorFactorSet** – Sets the interpolation factor for position compare mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must be not a secondary positioner): (-8)
- Check the encoder type (must be “AnalogInterpolated”): (-8)
- Check input parameter value: (-17)
- Check the positioner name: (-18)
- Check the group state (must be NOTINIT): (-22)

#### **Description**

This function sets the interpolation factor of the hardware interpolator used in the “PositionCompare” mode. The IP200 is updated and the position compare resolution is set as follows:

PositionCompareResolution = EncoderScalePitch/InterpolationFactor

The “InterpolationFactor“ value must be define with one of these values:

20	25	40	50	80	100	160	200
----	----	----	----	----	-----	-----	-----

If the input interpolator factor value is different from these values then (-17) error is returned.

---

#### **NOTE**

**The group must be NOTINIT to use this function else (-22) error is returned.**

**The encoder type must be “AnalogInterpolated” in the stages.ini file (“EncoderType” parameter) else the error is returned**

---

#### **Prototype**

```
int PositionerHardInterpolatorFactorSet(
    int SocketID,
    char * FullPositionerName,
    int * InterpolationFactor
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
InterpolationFactor	int	interpolation factor.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.251 PositionerHardInterpolatorPositionGet

#### Name

**PositionerHardInterpolatorPositionGet** – Gets interpolated position from the encoder hard interpolator.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must not be a secondary positioner): (-8)
- Check the encoder type (must be “AnalogInterpolated”): (-8)

#### Description

This function returns the position interpolated by the encoder hard interpolator.

---

#### NOTE

The encoder type must be “*AnalogInterpolated*” or “*AnalogInterpolatedTheta*” in the stages.ini file (“EncoderType” parameter).

---

#### Prototype

```
int PositionerHardInterpolatorPositionGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * Position  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

Position	double *	Interpolated position.
----------	----------	------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.252 PositionerHardwareStatusGet

#### Name

**PositionerHardwareStatusGet** – Gets the positioner hardware status code.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must be not a secondary positioner): (-8), (-18), (-24)

#### Description

This function returns the hardware status of the selected positioner. The positioner hardware status is composed of the “corrector” hardware status and the “servitudes” hardware status:

The “Corrector” returns the motor interface and the position encoder hardware status.

The “Servitudes” returns the general inhibit and the end of runs hardware status.

---

#### NOTE

See section 7.6: “Positioner Hardware Status List”.

---

#### Prototype

```
int PositionerHardwareStatusGet(  
    int SocketID,  
    char * FullPositionerName,  
    int * PositionerHardwareStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

PositionerHardwareStatus	int *	Hardware status code.
--------------------------	-------	-----------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.

### 6.2.1.253 PositionerHardwareStatusStringGet

#### Name

**PositionerHardwareStatusStringGet** – Gets the positioner error description.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the hardware status description from a positioner hardware status code.

---

#### NOTE

See section 7.6: “Positioner Hardware Status List”.

---

#### Prototype

```
int PositionerHardwareStatusStringGet(  
    int SocketID,  
    char * FullPositionerName,  
    int PositionerHardwareStatusCode,  
    char * PositionerHardwareStatusString  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
PositionerHardwareStatusCode	int	Positioner hardware status code.

#### Output parameters

PositionerHardwareStatusString	int *	Positioner hardware status description.
--------------------------------	-------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### **6.2.1.254 PositionerJogMaximumVelocityAndAccelerationGet**

#### **Name**

**PositionerJogMaximumVelocityAndAccelerationGet** – Gets the jog maximum velocity and jog maximum acceleration from profiler generator.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name: (-18)

#### **Description**

This function returns the jog maximum velocity and jog maximum acceleration of the profile generators. These parameters represent the limits for the profiler and are defined in the stages.ini file:

JogMaximumVelocity = ; unit/second  
 JogMaximumAcceleration = ; unit/second<sup>2</sup>

#### **Prototype**

```
int PositionerJogMaximumVelocityAndAccelerationGet(
    int SocketID,
    char * FullPositionerName,
    double * JogMaximumVelocity,
    double * JogMaximumAcceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

JogMaximumVelocity	double *	Jog maximum velocity (units/s).
JogMaximumAcceleration	double *	Jog maximum acceleration (units/s <sup>2</sup> ).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.

### 6.2.1.255 PositionerMagneticTrackPositionAtHomeGet

#### Name

**PositionerMagneticTrackPositionAtHomeGet** – Gets magnetic track position at home in units.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)

#### Description

This function returns the system compensation parameters defined for the second order low-pass filter.

#### Prototype

```
int PositionerMagneticTrackPositionAtHomeGet(  
    int SocketID,  
    char * PositionerName,  
    double * MagneticTrackPosition  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

MagneticTrackPosition	double *	magnetic track position at home (units).
-----------------------	----------	--

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.256 PositionerMaximumVelocityAndAccelerationGet

#### Name

**PositionerMaximumVelocityAndAccelerationGet** – Gets the maximum velocity and acceleration from the profiler generators.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must not be a secondary positioner): (-8)

#### Description

This function returns the maximum velocity and acceleration of the profile generators. These parameters represent the limits for the profiler and are defined in the stages.ini file:

MaximumVelocity =; unit/second

MaximumAcceleration = ; unit/second<sup>2</sup>

#### Prototype

```
int PositionerMaximumVelocityAndAccelerationGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * MaximumVelocity,  
    double * MaximumAcceleration  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

MaximumVelocity	double *	Maximum velocity (units/seconds).
MaximumAcceleration	double *	Maximum acceleration (units/seconds <sup>2</sup> ).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.257 PositionerMotionDoneGet**

#### **Name**

**PositionerMotionDoneGet** – Gets the motion done parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must not be a secondary positioner): (-8)
- Check the motion done mode (must be “VelocityAndPositionWindow”): (-8)

#### **Description**

This function returns the motion done parameters only for the “VelocityAndPositionWindow” MotionDone mode. If the MotionDone mode is defined as “Theoretical” then (-8) error is returned.

The “MotionDoneMode” parameter from the stages.ini file defines the motion done mode. The motion done can be defined as “Theoretical” (the motion done mode is not used) or “VelocityAndPositionWindow”. For a more thorough description of the motion done mode, please refer to the XPS Motion Tutorial section Motion/Motion Done.

#### **Prototype**

```
int PositionerMotionDoneGet(
    int SocketID,
    char * FullPositionerName,
    double * PositionWindow,
    double * VelocityWindow,
    double * MeanPeriod
    double * Timeout
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

PositionWindow	double *	Position window (units).
VelocityWindow	double *	Velocity window (units/seconds).
CheckingTime	double *	Checking time (seconds).
MeanPeriod	double *	Mean period (seconds).
Timeout	double *	Motion done time out (seconds).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.258 PositionerMotionDoneSet**

#### **Name**

**PositionerMotionDoneSet** – Sets the motion done parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check parameter value: (-17)

#### **Description**

This function updates the motion done parameters only for the “VelocityAndPositionWindow” MotionDone mode. The “MotionDoneMode” parameter from the stages.ini file must be defined as “VelocityAndPositionWindow” else (-8) error is returned.

For a more thorough description of the Motion Done mode, please refer to the XPS Motion Tutorial section Motion/Motion Done.

#### **Prototype**

```
int PositionerMotionDoneSet(
    int SocketID,
    char * FullPositionerName,
    double PositionWindow,
    double VelocityWindow,
    double MeanPeriod
    double Timeout
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
PositionWindow	double	Position window (units).
VelocityWindow	double	Velocity window (units/seconds).
CheckingTime	double	Checking time (seconds).
MeanPeriod	double	Mean period (seconds).
Timeout	double	Motion done time out (seconds).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.259 PositionerPositionCompareAquadBAlwaysEnable**

#### **Name**

**PositionerPositionCompareAquadBAlwaysEnable** – Enables the AquadB signal in the always mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check if the CIE board supports this function: (-115)

#### **Description**

This function enables the generation of AquadB output signals on the PCO connector (the 2&3 or 4&5 pins) of the XPS controller cards. The “always” mode means that the AquadB signal is generated all the time (not position windowed).

---

#### **NOTE**

**This function can be used only with a position encoder. If there is no position encoder then (-24) error is returned.**

---

#### **Prototype**

```
int PositionerPositionCompareAquadBAlwaysEnable(  
    int SocketID,  
    char * FullPositionerName  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -24: Not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware.

### **6.2.1.260 PositionerPositionCompareAquadBPrescalerGet**

#### **Name**

**PositionerPositionCompareAquadBPrescalerGet** – Gets PCO AquadB interpolation factor.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check encoder type (must be “AnalogSinCos” encoder): (-8)
- Check parameter value: (-17)
- Check positioner name: (-18)

#### **Description**

This function gets the current Position Compare AquadB interpolation factor.

The predefined PCO interpolation values are the following:

- 4.
- 16.
- 256.
- 4096.
- 65536.

#### **Prototype**

```
int PositionerPositionCompareAquadBPrescalerGet(  
    int SocketID,  
    char PositionerName,  
    double * PCOInterpolationFactor  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the TCP_ConnectToServer” function.
PositionerName	int	Positioner name.

#### **Output parameters**

PCOInterpolationFactor	double *	Current PCO interpolation factor.
------------------------	----------	-----------------------------------

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner name doesn't exist or unknown command.

### **6.2.1.261 PositionerPositionCompareAquadBPrescalerSet**

#### **Name**

**PositionerPositionCompareAquadBPrescalerSet** – Sets PCO AquadB interpolation factor.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check encoder type (must be “AnalogSinCos” encoder): (-8)
- Check PCO interpolation factor value: (-17)
- Check positioner name: (-18)

#### **Description**

This function configures Position Compare AquadB interpolation factor.

The predefined PCO interpolation values are the following:

- 4
- 16
- 256
- 4096
- 65536

#### **Prototype**

```
int PositionerPositionCompareAquadBPrescalerSet(
    int SocketID,
    char PositionerName,
    double PCOInterpolationFactor
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the TCP_ConnectToServer” function.
PositionerName	int	Positioner name.
PCOInterpolationFactor	double	Predefined PCO interpolation factor value.

#### **Output parameters**

None.

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.262 PositionerPositionCompareAquadBWindowedGet**

#### **Name**

**PositionerPositionCompareAquadBWindowedGet** – Gets the windowed AquadB mode parameters and state..

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check the position compare parameters (must be configured): (-23)
- Check the configured mode type (must be WindowedAquadB): (-24)
- Check if the CIE board supports this function: (-115)

#### **Description**

This function returns the configured parameters of the position windowed AquadB output signal and gives its state (enabled or disabled).

#### **NOTE**

**This function can be used only with a position encoder (“AquadB” or “AnalogInterpolated”). If there is no position encoder then (-24) error is returned.**

#### **Prototype**

```
int PositionerPositionCompareAquadBWindowedGet (int SocketID, char
FullPositionerName[250] , double* MinimumPosition, double* MaximumPosition, bool
* EnableState)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
EnableState	bool *	Windowed AquadB state (true=enabled or false=disabled)

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -23: ERR\_POSITION\_COMPARE\_NOT\_SET.
- -24: Not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware.

### **6.2.1.263 PositionerPositionCompareAquadBWindowedSet**

#### **Name**

**PositionerPositionCompareAquadBWindowedSet** – Sets the windowed AquadB signal parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check input parameter values: (-17)
  - MinimumPosition < MaximumPosition.
  - MinimumPosition  $\geq$  MinimumTargetPosition.
  - MaximumPosition  $\leq$  MaximumTargetPosition.
- Check position compare state (must be disabled): (-22)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check if the CIE board supports this function: (-115)

#### **Description**

This function sets the parameters for the position windowed AquadB output signal on the PCO connector (the 2&3 or 4&5 pins) of the XPS controller cards.

These parameters are in effect only when the position compare mode is enabled by the *PositionerPositionCompareEnable()* function.

#### **NOTE**

**This function can be used only with a position encoder (“AquadB” or “AnalogInterpolated”). If there is no position encoder then (-24) error is returned.**

#### **Prototype**

```
int PositionerPositionCompareAquadBWindowedSet(
    int SocketID,
    char * FullPositionerName
    double MinimumPosition,
    double MaximumPosition
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
MinimumPosition	double	Minimum position (units).
MaximumPosition	double	Maximum position (units).

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware.

### 6.2.1.264 PositionerPositionCompareDisable

#### Name

**PositionerPositionCompareDisable** – Disables the position compare mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must not be a secondary positioner): (-8)
- Check the encoder (“AquadB” or “AnalogInterpolated”): (-8)

#### Description

This function disables the position compare mode.

For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

#### Prototype

```
int PositionerPositionCompareDisable(
```

```
    int SocketID,  
    char * FullPositionerName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.265 PositionerPositionCompareEnable**

#### **Name**

**PositionerPositionCompareEnable** – Enables the position compare mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must not be a secondary positioner): (-8)
- Check the encoder (“AquadB” or “AnalogInterpolated”): (-8)
- Check the positioner name: (-18)
- Check the group state (must be READY): (-22)
- Check the position compare parameters (must be configured): (-22)

#### **Description**

This function enables the position compare mode. The group must be in READY state to use this function else (-22) error is returned.

If the position compare parameters are not configured (by the “PositionerPositionCompareSet” function) then (-22) error is returned.

For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

#### **Prototype**

```
int PositionerPositionCompareEnable(  
    int SocketID,  
    char * FullPositionerName  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -22: Not allowed action.

### **6.2.1.266 PositionerPositionCompareGet**

#### **Name**

**PositionerPositionCompareGet** – Gets the position compare parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must not be a secondary positioner): (-8)
- Check the position compare parameters (must be configured): (-23)
- Check the configured mode type (must be PositionCompare): (-24)

#### **Description**

This function returns the real value (without correction) of parameters of the position compare output trigger and returns current state (enabled or disabled).

For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

#### **Prototype**

```
int PositionerPositionCompareGet(
    int SocketID,
    char * FullPositionerName,
    double * MinimumPosition,
    double * MaximumPosition,
    double * PositionStep,
    bool * EnableState
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets b the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
PositionStep	double *	Position step (units).
EnableState	bool *	Position compare state (true = enabled or false = disabled).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -23: Position compare not set.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.267 PositionerPositionComparePulseParametersGet

#### Name

**PositionerPositionComparePulseParametersGet** – Gets the position compare PCO pulse parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner (must not be a secondary positioner): (-8)

#### Description

This function returns the configured parameters of the position compare PCO pulse parameters.

For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

#### Prototype

```
int PositionerPositionComparePulseParametersGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * PCOPulseWidth,  
    double * EncoderSettlingTime  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

PCOPulseWidth	double *	Width of PCO pulses ( $\mu$ s).
EncoderSettlingTime	double *	Encoder signal settling time ( $\mu$ s).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.268 PositionerPositionComparePulseParametersSet

#### Name

**PositionerPositionComparePulseParametersSet** – Sets the position compare PCO pulse parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)
- Check input parameter values: (-17)
  - PCOPulseWidth value must equal to 0.2 (default), 1, 2.5 or 10 ( $\mu$ s)
  - EncoderSettlingTime value must equal to 0.075 (default), 1, 4 or 12 ( $\mu$ s)
- Check position compare state (must be disabled): (-22)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Check if the CIE board supports this function: (-115)

#### Description

This function sets two additional parameters for the position compare output trigger of the PCO connector on the XPS controller cards. The first additional parameter is the pulse width. The second parameter is the encoder settling time value, which is the time the encoder inputs have to stabilize after a change of state is detected.

These parameters are used only when using the position compare mode. For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

---

#### NOTE

**When changing the PCO Pulse settle time you must limit the maximum velocity of the stage accordingly, otherwise you will lose the PCO position and generate the wrong number of pulses at wrong positions.**

**For example, if you set the EncoderSettlingTime to 4  $\mu$ s, the maximum PCO encoder frequency need to be limited to less than  $0.25 / 4e^{-6} = 62.5$  kHz.**

**So, if EncoderScalePitch = 0.004 mm and HardInterpolatorFactor = 200 then the stage maximum velocity must  $\leq 62.5e^3 * 0.004 / 200 = 1.25$  mm/s, otherwise the PCO will not work properly.**

---

How to determine PCO encoder frequency:

1. For AquadB encoder:

$$\text{PCO encoder frequency} = \text{Velocity} / \text{EncoderResolution}$$

2. For analog interpolated encoder:

$$\text{PCO encoder frequency} = \text{Velocity} * \text{HardInterpolatorFactor} / \text{EncoderScalePitch}$$

**Example:** XML310 stage ( $\text{EncoderScalePitch} = 0.004$  mm,  
 $\text{HardInterpolatorFactor} = 200$ ). If  $\text{Velocity} = 10\text{mm/s} = >\text{PCO encoder frequency} = 10 * 200/0.004 = 500$  kHz

---

**NOTE**

This function can be used only with a position encoder. If no position encoder then (-24) error is returned.

This function is called automatically at controller reboot and at GroupInitialize() execution to set the position compare pulse parameters to default values (PCOPulseWidth to 0.2 µs, EncoderSettlingTime to 0.075 µs).

---

**Prototype**

```
int PositionerPositionComparePulseParametersSet(
```

```
    int SocketID,  
    char * FullPositionerName,  
    double PCOPulseWidth,  
    double EncoderSettlingTime  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
PCOPulseWidth	double	Width of PCO pulses (µs).
EncoderSettlingTime	double	Encoder signal settling time (µs).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware.

### **6.2.1.269 PositionerPositionCompareScanAccelerationLimitGet**

#### **Name**

**PositionerPositionCompareScanAccelerationLimitGet** – Gets the position compare scan acceleration limit.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the corrector type (must be *PIDFFAcceleration* corrector): (-24)

#### **Description**

This function returns the position compare scan acceleration limit.

During scan of position compare, the motor output will be limited to this value instead of the *AccelerationLimit*.

The position compare scan acceleration limit takes effect only with *PIDFFAcceleration* corrector type.

This function can be used only with a *PIDFFAcceleration* corrector else (-24) error is returned.

For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

#### **Prototype**

```
int PositionerPositionCompareScanAccelerationLimitGet(
```

```
    int SocketID,  
    char * FullPositionerName,  
    double * ScanAccelerationLimit  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

ScanAccelerationLimit	double *	Limit of position compare scan acceleration (units/s <sup>2</sup> ).
-----------------------	----------	---

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware.

### **6.2.1.270 PositionerPositionCompareScanAccelerationLimitSet**

#### **Name**

**PositionerPositionCompareScanAccelerationLimitSet** – Sets the position compare acceleration limit.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter values: (-17)
  - PositionCompareScanAccelerationLimit value must >0 and ≤MaximumAcceleration (value in stages.ini)
- Check the corrector type (must be *PIDFFAcceleration* corrector): (-24)

#### **Description**

This function sets the position compare scan acceleration limit.

During position compare, the motor output will be limited to this value instead of *AccelerationLimit*.

The position compare scan acceleration limit takes effect only with *PIDFFAcceleration* corrector type.

This function can be used only with a *PIDFFAcceleration* corrector otherwise *ERR\_UNCOMPATIBLE* is returned.

For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

#### **Prototype**

```
int PositionerPositionCompareScanAccelerationLimitSet(  
    int SocketID,  
    char * FullPositionerName,  
    double * ScanAccelerationLimit  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
ScanAccelerationLimit	double	Limit of position compare scan acceleration (units/s <sup>2</sup> ).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.271 PositionerPositionCompareSet

#### Name

**PositionerPositionCompareSet** – Sets the position compare parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type: (-8)
- Check input parameter values: (-17)
  - MinimumPosition < MaximumPosition.
  - MinimumPosition ≥ MinimumTargetPosition.
  - MaximumPosition ≤ MaximumTargetPosition.
  - $0 \leq \text{PositionStep} \leq (\text{MaximumPosition} - \text{MinimumPosition})$
- Check position compare state (must be disabled): (-22)
- Check the position encoder (“AquadB” or “AnalogInterpolated”): (-24)

#### Description

This function sets the parameters for the position compare output trigger of the PCO connector on the XPS controller cards.

These parameters are used only when the position compare mode is enabled. For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

---

#### NOTE

**This function can be used only with a position encoder. If no position encoder then (-24) error is returned.**

**In the PositionCompare mode (activated with PositionerPositionCompareEnable() function), during the move (relative or absolute) and inside the zone set by PositionerPositionCompareSet(), if the current following error exceeds the Warning Following Error value, the PositionCompareWarningFollowingErrorFlag is activated and the move returns an error (-120: Warning following error during move with position compare enabled). To reset the PositionCompareWarningFollowingErrorFlag, send the PositionerPositionCompareDisable() function. The WarningFollowingError is set to FatalFollowingError (defined in stages.ini file) by default, but it can be modified by PositionerWarningErrorSet().**

**In the PositionCompare mode (activated with PositionerPositionCompareEnable() function), during the move (relative or absolute) and inside the zone set by PositionerPositionCompareSet(), the CorrectorOutput is limited to MaximumAcceleration (*defined in stages.ini*).**

---

### **Prototype**

```
int PositionerPositionCompareSet(  
    int SocketID,  
    char * FullPositionerName,  
    double MinimumPosition,  
    double MaximumPosition,  
    double PositionStep,  
    bool EnableState  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
MinimumPosition	double	Minimum position (units).
MaximumPosition	double	Maximum position (units).
PositionStep	double	Position step (units).
EnableState	bool	Position compare state (true = enabled or false = disabled).

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.272 PositionerPreCorrectorExcitationSignalGet

#### Name

**PositionerPreCorrectorExcitationSignalGet** – Returns the currently used parameters of the pre-corrector excitation signal feature.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Valid positioner name: (-18)

#### Description

This function gets the last configured pre-corrector excitation signal parameters.

#### Prototype

```
int PositionerPreCorrectorExcitationSignalGet(
```

```
    int SocketID,  
    char * FullPositionerName  
    double * Frequency,  
    double * Amplitude,  
    double * Time  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

Frequency	double *	Frequency (Hz).
Amplitude	double *	Amplitude (position units).
Time	double *	During time (seconds).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.

### **6.2.1.273 PositionerPreCorrectorExcitationSignalSet**

#### **Name**

**PositionerPreCorrectorExcitationSignalSet** – Configure and activate the signal of pre-corrector excitation.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Is secondary positioner or has a secondary positioner: (-8)
- Check frequency (must  $\geq 0.1$  and  $\leq 0.5/\text{CorrectorISRPeriod}$ ): (-17)
- Check amplitude (*must >0*): (-17)
- Check excitation time (*must  $\geq 4 * \text{CorrectorISRPeriod}$* ): (-17)
- Valid positioner name: (-18)
- Check group status (*must be READY*): (-22)
- Valid control loop type: (-24)
- Check position ( $\text{CurrSetpointPos} \leq \text{UserMaximumTargetPosition}$  and  $\text{CurrSetpointPos} - 2\text{Amplitude} \geq \text{UserMinimumTargetPosition}$ ): (-35)
- Check maximum velocity ( $\text{Amplitude} * \omega < \text{MaximumVelocity}$ ): (-68)
- Check maximum acceleration ( $\text{Amplitude} * \omega^2 < \text{MaximumAcceleration}$ ): (-69)

#### **Description**

The pre-corrector excitation signal functionality generates sine-wave signals (*ExcitationPosition*, *ExcitationVelocity*, *ExcitationAcceleration*, *ExcitationJerk*) inserted in the position control loop (*in closed or open loop configuration*) to excite the system. In measuring the output signal (*e.g. encoder position, velocity or acceleration*) of the excited system, we can determine some system characteristics, like the system transfer function.

The exact forms of generated pre-corrector excitation signals are as follows:

$$\omega = 2\pi F \quad (F: \text{excitation frequency})$$

$$\text{ExcitationPosition} = A * \cos(\omega t) - A \quad (A: \text{excitation amplitude}, t: \text{current time})$$

$$\text{ExcitationVelocity} = (-A\omega) * \sin(\omega t)$$

$$\text{ExcitationAcceleration} = (-A\omega^2) * \cos(\omega t)$$

$$\text{ExcitationJerk} = (A\omega^3) * \sin(\omega t)$$

The pre-corrector excitation signal functionality is only available with the stages controlled in acceleration (acceleration control, ex: brushless / linear motors), velocity (velocity control) or in voltage (voltage control). It does not exist with stages controlled in position (ex: stepper motors).

The excitation signal function **PositionerPreCorrectorExcitationSignalSet** can be executed only when the positioner is in “READY” state. When the excitation-signal process is in progression, the positioner is in the “ExcitationSignal” state. At the end of the process, the positioner returns to the “READY” state (see group state diagram).

The **PositionerPreCorrectorExcitationSignalSet** function sends a sine form excitation signal to entry of position control loop during a time. This function is allowed for “PIDFFAcceleration”, “PIDFFVelocity” or “PIDDualFFVoltage” control loops. The parameters to configure include: *frequency* (Hz, double), *amplitude* (position unit, double) and *during time* (seconds, double).

The function effective parameters are: *Frequency* ( $\geq 0.1$  and  $\leq 0.5 / \text{CorrectorISRPeriod}$ ), *Amplitude* ( $> 0$ ), *Time* ( $\geq 4 * \text{CorrectorISRPeriod}$ ).

---

#### NOTE

**If during the excitation signal generation the stage position exceeds the user minimum or maximum target positions, the motor excitation command is stopped and an error is returned.**

---

#### Prototype

```
int PositionerPreCorrectorExcitationSignalSet(
```

```
    int SocketID,
    char * FullPositionerName
    double Frequency,
    double Amplitude,
    double Time
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
Frequency	double	Frequency (Hz).
Amplitude	double	Amplitude (position units).
Time	double	During time (seconds).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).
- -35: Position is outside of travel limits.
- -68: Velocity on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -112: Error of excitation signal generation initialization.

### 6.2.1.274 PositionerRawEncoderPositionGet

#### Name

**PositionerRawEncoderPositionGet** – Returns the raw encoder position for a positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Valid positioner name: (-18)

#### Description

This function returns the raw encoder position from a corrected position for a positioner.

#### Prototype

```
int PositionerRawEncoderPositionGet(  
    int SocketID,  
    char * FullPositionerName  
    double UserEncoderPosition,  
    double * RawEncoderPosition  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
UserEncoderPosition	double	User corrected encoder position.

#### Output parameters

RawEncoderPosition      double \*      Raw encoder position.

#### Return (In addition to the results of testing the inputs of functions)

- 0:      No error.
- -8:      Wrong object type for this command.
- -18:      Positioner Name doesn't exist or unknown command.

### **6.2.1.275 PositionerSGammaExactVelocityAjustedDisplacementGet**

#### **Name**

**PositionerSGammaExactVelocityAjustedDisplacementGet** – Gets the adjusted displacement to get exact velocity.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the profiler type (must be “SGamma”): (-8)
- Check the positioner type (must not be a secondary positioner): (-8)

#### **Description**

This function returns the closest optimum displacement to obtain the most precise velocity during the displacement.

#### **Prototype**

```
int PositionerSGammaExactVelocityAjustedDisplacementGet(  
    int SocketID,  
    char * FullPositionerName  
    double DesiredDisplacement,  
    double * AdjustedDisplacement  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
DesiredDisplacement	double	Desired displacement (units).

#### **Output parameters**

AdjustedDisplacement	double *	Ajusted displacement (units).
----------------------	----------	-------------------------------

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.276 PositionerSGammaParametersGet

#### Name

**PositionerSGammaParametersGet** – Gets the current motion values from the SGamma profiler.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the profiler type (must be “SGamma”): (-8)
- Check the positioner type (must not be a secondary positioner): (-8)

#### Description

This function gets the current SGamma profiler values that are used in displacements.

#### Prototype

```
int PositionerSGammaParametersGet(  
    int SocketID,  
    char * FullPositionerName  
    double * Velocity,  
    double * Acceleration,  
    double * MinimumJerkTime,  
    double * MaximumJerkTime  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

Velocity	double *	motion velocity (units/seconds).
Acceleration	double *	motion acceleration (units/seconds <sup>2</sup> ).
MinimumJerkTime	double *	Minimum jerk time (seconds).
MaximumJerkTime	double *	Maximum jerk time (seconds).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.277 PositionerSGammaParametersSet**

#### **Name**

**PositionerSGammaParametersSet** – Sets new motion values for the SGamma profiler.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the profiler type (must be “SGamma”): (-8)
- Check the positioner type (must not be a secondary positioner): (-8)
- Check input parameter values: (-17)
  - $0 < \text{NewVelocity} \leq \text{MaximumVelocity}$ .
  - $0 < \text{NewAcceleration} \leq \text{MaximumAcceleration}$ .
  - $2 * \text{ISRProfileGeneratorPeriod} \leq \text{NewMinimumJerkTime}$  and  $\leq \text{NewMaximumJerkTime}$ .

#### **Description**

This function defines the new SGamma profiler values that will be used in future displacements.

#### **Prototype**

```
int PositionerSGammaParametersSet(
    int SocketID,
    char * FullPositionerName
    double Velocity,
    double Acceleration,
    double MinimumJerkTime,
    double MaximumJerkTime
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
Velocity	double	motion velocity (units/seconds).
Acceleration	double	motion acceleration (units/seconds <sup>2</sup> ).
MinimumJerkTime	double	Minimum jerk time (seconds).
MaximumJerkTime	double	Maximum jerk time (seconds).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.

### **6.2.1.278 PositionerSGammaPreviousMotionTimesGet**

#### **Name**

**PositionerSGammaPreviousMotionTimesGet** – Gets the motion and the settling time.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the profiler type (must be “SGamma”): (-8)
- Check the positioner type (must not be a secondary positioner): (-8)

#### **Description**

This function returns the motion (setting) and settling times from the previous motion.  
The motion time represents the defined time to complete the previous displacement.  
The settling time represents the effective settling time for a motion done.

#### **Description**

This function gets the current SGamma profiler values that are used in displacements.

#### **Prototype**

```
int PositionerSGammaPreviousMotionTimesGet(
    int SocketID,
    char * FullPositionerName
    double * SettingTime,
    double * SettlingTime
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

SettingTime	double *	Setting time (seconds).
SettlingTime	double *	Settling time (seconds).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.279 PositionerSGammaVelocityAndAccelerationSet

#### Name

**PositionerSGammaVelocityAndAccelerationSet** – Sets the SGamma profiler velocity and acceleration.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter values: (-17)
  - $0 < \text{NewVelocity} \leq \text{MaximumVelocity}$
  - $0 < \text{NewAcceleration} \leq \text{MaximumAcceleration}$

#### Description

This function sets the SGamma profiler velocity and acceleration that will be used for the future displacements.

#### Prototype

```
int PositionerSGammaVelocityAndAccelerationSet(  
    int SocketID,  
    double Velocity,  
    double Acceleration  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Velocity	double	motion velocity (units / seconds).
Acceleration	double	motion acceleration (units / seconds <sup>2</sup> ).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.1.280 PositionerStageParameterGet

#### Name

**PositionerStageParameterGet** – Gets a stage parameter value from the stages.ini file.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name and the parameter name: (-24)

#### Description

This function returns stage parameter values from the stages.ini file of a selected positioner.

The positioner name is the stage name. And the parameter name is read in the section under this stage name.

#### Prototype

```
int PositionerStageParameterGet(  
    int SocketID,  
    char * FullPositionerName  
    char * ParameterName,  
    char * ParameterValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
ParameterName	char *	Parameter name.

#### Output parameters

ParameterValue	char *	Parameter value.
----------------	--------	------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -13: Wrong parameter type in the command string: char \* expected.
- -24: Not available in this configuration (check hardware or configuration).

### **6.2.1.281 PositionerStageParameterSet**

#### **Name**

**PositionerStageParameterSet** – Saves a stage parameter value into the stages.ini file.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name and the parameter name: (-24)
- Check the user rights (must be identified as administrator): (-107)

#### **Description**

This function saves a stage parameter value in the “stages.ini” file.

The positioner name sets the stage name and the parameter name is searched in the section of this stage name. Once the parameter is found, the parameter value is modified to the new value.

If file reading fails then (-61) error is returned

If file writing fails then (-60) error is returned

#### **NOTE**

**To use this function, the user must login with administrator rights (“Login” function).**

#### **Prototype**

```
int PositionerStageParameterSet(
    int SocketID,
    char * FullPositionerName
    char * ParameterName,
    char * ParameterValue
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
ParameterName	char *	Parameter name.

#### **Output parameters**

ParameterValue	char *	Parameter value.
----------------	--------	------------------

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -24: Not available in this configuration (check hardware or configuration).
- -60: Error during file writing or file doesn't exist.
- -61: Error file corrupt or file doesn't exist.
- -107: This function requires to be logged in with Administrator rights.

### 6.2.1.282 PositionerTimeFlasherDisable

#### Name

**PositionerTimeFlasherDisable** – Disables the time flasher mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)

#### Description

This function disables the time flasher mode. The time flasher mode is a trigger output per axis that can be either configured to output distance spaced pulses or time spaced pulses. The output pulses are accessible from the PCO connector at the back of the XPS controller.

For a more thorough description of the position compare output, please refer to the XPS User’s manual, section named Triggers/Position Compare Output.

#### Prototype

```
int PositionerTimeFlasherDisable(  
    int SocketID,  
    char * FullPositionerName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.1.283 PositionerTimeFlasherEnable

#### Name

**PositionerTimeFlasherEnable** – Enables the time flasher mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)
- Check the time flasher parameters (must be configured): (-22)

#### Description

This function enables the time flasher mode. The time flasher mode is a trigger output per axis that can be either configured to output distance spaced pulses or time spaced pulses. The output pulses are accessible from the PCO connector at the back of the XPS controller.

To use this function, the group must be in READY state else (-22) error is returned.

For a more thorough description of the position compare output, please refer to the XPS User’s manual, section named Triggers/Position Compare Output.

#### Prototype

```
int PositionerTimeFlasherEnable(  
    int SocketID,  
    char * FullPositionerName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### **6.2.1.284 PositionerTimeFlasherGet**

#### **Name**

**PositionerTimeFlasherGet** – Gets the time flasher parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)
- Check the time flasher parameters (must be configured): (-23)
- Check the configured mode type (must be TimeFlasher): (-24)

#### **Description**

This function returns the parameters of the time flasher trigger. The time flasher mode is defined by:

a position window defined by a minimum position and a maximum position  
a time period to set the time spaced pulses.

For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.

#### **Prototype**

```
int PositionerTimeFlasherGet(
```

```
    int SocketID,
    char * FullPositionerName,
    double * MinimumPosition,
    double * MaximumPosition,
    double * TimePeriod,
    bool * EnableState
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
TimePeriod	double *	Time period (seconds).
EnableState	bool *	Enable time flasher state (true = enabled and false = disabled).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -23: Position compare not set.
- -24: Not available in this configuration (check hardware or configuration).

### 6.2.1.285 PositionerTimeFlasherSet

#### Name

**PositionerTimeFlasherSet** – Sets the time flasher parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)
- Check input parameter values: (-17)
  - MinimumPosition < MaximumPosition.
  - MinimumPosition ≥ MinimumTravelLimit.
  - MaximumPosition ≤ MaximumTravelLimit.
  - 0.0000004 ≤ TimePeriod ≤ 50.0 (Max 2.5 MHz and Min 0.02 Hz)
- Check the time flasher state (must be disabled): (-22)
- Check the position encoder (must be used): (-24)
- Check if the CIE board supports this function: (-115)

#### Description

This function configures the time flasher parameters. The time flasher output trigger uses the PCO connector on the XPS controller cards. The time flasher mode is defined by:

- a position window defined by a minimum position and a maximum position.
- a time period to set the time spaced pulses.

---

#### NOTES

**This function is not available without a position encoder.**

**These parameters are used only when the time flasher mode is enabled. To enable the time flasher mode, use the “PositionerPositionCompareEnable” function.**

**For a more thorough description of the position compare output, please refer to the XPS Motion Tutorial section Triggers/Position Compare Output.**

---

#### Prototype

```
int PositionerTimeFlasherSet(  
    int SocketID,  
    char * FullPositionerName,  
    double MinimumPosition,  
    double MaximumPosition,  
    double TimePeriod,  
    bool EnableState  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
MinimumPosition	double	Minimum position (units).
MaximumPosition	double	Maximum position (units).
TimePeriod	double	Time period (seconds).
EnableState	bool	Enable time flasher state (true = enabled and false = disabled).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.
- -24: Not available in this configuration (check hardware or configuration).
- -115: Function is not supported by current hardware.

### 6.2.1.286 PositionerUserTravelLimitsGet

#### Name

**PositionerUserTravelLimitsGet** – Gets the user travel limits.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)
- If piezo driver, check if driver is not initialized: (-118)

#### Description

This function returns the user-defined travel limits for the selected positioner.

#### Prototype

```
int PositionerUserTravelLimitsGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * UserMinimumTarget,  
    double * UserMaximumTarget  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### Output parameters

UserMinimumTarget	double *	User minimum travel limit (units).
UserMaximumTarget	double *	User maximum travel limit (units).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -118: Not allowed action driver not initialized.

### **6.2.1.287 PositionerUserTravelLimitsSet**

#### **Name**

**PositionerUserTravelLimitsSet** – Sets the user travel limits.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner type (must not be a secondary positioner): (-8)
- Check input parameter values: (-17)
  - UserMinimumTargetPosition <UserMaximumTargetPosition.
  - MinimumTargetPosition ≤UserMinimumTargetPosition ≤MaximumTargetPosition.
  - MinimumTargetPosition ≤UserMaximumTargetPosition ≤MaximumTargetPosition.
  - UserMinimumTargetPosition ≤ProfilerPosition.
  - UserMaximumTargetPosition ≥ProfilerPosition.
- If piezo driver, check if driver is not initialized: (-118)

#### **Description**

This function sets the new user travel limits of the selected positioner.

#### **Prototype**

```
int PositionerUserTravelLimitsSet(
    int SocketID,
    char * FullPositionerName,
    double UserMinimumTarget,
    double UserMaximumTarget
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
UserMinimumTarget	double	User minimum travel limit (units).
UserMaximumTarget	double	User maximum travel limit (units).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Wrong parameter type in the command string: double or double \* expected.
- -118: Not allowed action driver not initialized.

### **6.2.1.288 PositionerWarningFollowingErrorGet**

#### **Name**

**PositionerWarningFollowingErrorGet** – Returns the warning following error for a positioner.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Valid positioner name: (-18)

#### **Description**

This function gets the current value of the warning following error for a positioner.

#### **Prototype**

```
int PositionerWarningFollowingErrorGet(  
    int SocketID,  
    char * FullPositionerName,  
    double * WarningFollowingError  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.

#### **Output parameters**

WarningFollowingError    double \*    Warning following error (units).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0:         No error.
- -8:        Wrong object type for this command.
- -18:      Positioner Name doesn't exist or unknown command.

### 6.2.1.289 PositionerWarningFollowingErrorSet

#### Name

**PositionerWarningFollowingErrorSet** – Sets value of the warning following error for a positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Check input parameter values: (-17)
  - $0 < \text{WarningFollowingError} \leq \text{FatalFollowingError}$ .
- Valid positioner name: (-18)

#### Description

This function sets a new value of the warning following error for a positioner.

#### Prototype

```
int PositionerWarningFollowingErrorSet(  
    int SocketID,  
    char * FullPositionerName,  
    double WarningFollowingError  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
FullPositionerName	char *	Positioner name.
WarningFollowingError	double	Warning following error (units).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner name doesn't exist or unknown command.

### 6.2.1.290 Reboot

#### Name

**Reboot** – Reboots the controller.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function reboots the controller.

---

#### NOTE

**This function is not a hardware reboot (power off/on), it is a Bios reboot.**

**If an FTP client is connected, this function is not allowed and (-22) error is returned.**

---

#### Prototype

```
int Reboot(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -22: Not allowed action.

### 6.2.1.291 RestartApplication

#### Name

**RestartApplication** – Restarts the controller's application and avoids hardware reboot.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".

#### Description

This function allows restarting controller applications without hardware reboot.

#### Prototype

```
int RestartApplication(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -22: Not allowed action.

### 6.2.1.292 RunPidin

#### Name

**RunPidin** – Saves the result of “pidin” command in a log file to get information about the processes in the system (QNX Neutrino).

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check error code returned by system shell command: (-100)

#### Description

This function executes “pidin” system command to get information about the processes in the system (QNX Neutrino) and save the result in the *pidin.log* file. This file is generated in *\Public\Log* folder.

**“pidin.log” extract example:**

pid	tid	name	prio	STATE	Blocked
1	1	/procnto-smp-instr	0f	RUNNING	
1	2	/procnto-smp-instr	0f	READY	
1	3	/procnto-smp-instr	255r	RECEIVE	1
1	4	/procnto-smp-instr	255r	RECEIVE	1
1	5	/procnto-smp-instr	65r	RECEIVE	1
1	6	/procnto-smp-instr	255r	RECEIVE	1
1	8	/procnto-smp-instr	10r	RECEIVE	1
1	9	/procnto-smp-instr	10r	RECEIVE	1
1	10	/procnto-smp-instr	10r	RECEIVE	1
1	11	/procnto-smp-instr	65r	RECEIVE	1
1	12	/procnto-smp-instr	65r	RECEIVE	1
1	14	/procnto-smp-instr	65r	RUNNING	
1	16	/procnto-smp-instr	65r	RECEIVE	1
2	1	proc/boot/devc-con	65r	RECEIVE	1
8195	1	proc/boot/pipe	10r	SIGWAITINFO	
8195	2	proc/boot/pipe	10r	RECEIVE	1
8195	3	proc/boot/pipe	10r	RECEIVE	1
8195	4	proc/boot/pipe	10r	RECEIVE	1
8195	5	proc/boot/pipe	10r	RECEIVE	1

#### Prototype

```
int RunPidin(
    int SocketID
)
int RunPidin(int SocketID, char *Options) // Only used in Terminal (expert)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Options	int	Options to add in “pidin” command.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -9: Wrong number of parameters in the command.
- -100: Internal error.

### 6.2.1.293 SingleAxisSlaveModeDisable

#### Name

**SingleAxisSlaveModeDisable** – Disables the slave-master mode.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the group type (must be a SingleAxis group): (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group state must be "SLAVE": (-22)

#### Description

This function disables the master-slave mode. If a motion is in progress then it is aborted.

To use this function, the group state must be SLAVE (46). If it's not the case then (-22) is returned.

#### Prototype

```
int SingleAxisSlaveModeDisable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
GroupName	char *	SingleAxis group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.294 SingleAxisSlaveModeEnable**

#### **Name**

**SingleAxisSlaveModeEnable** – Enables the slave-master mode.

#### **Input tests**

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the group type (must be a SingleAxis group): (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group state must be "READY": (-22)
- Check the slave parameters (must be configured): (-41)

#### **Description**

This function enables the master-slave mode only if the slave group is in "READY" state. In this mode the slave must be defined as a SingleAxis group and the master can be a positioner from any group.

To use this function, the SingleAxis group must be in the READY state. If it's not the case then (-22) error is returned.

To use this function, the master positioner and the slave ratio must be configured using the "SingleAxisSlaveParametersSet" function. If it's not the case then (-41) error is returned.

#### **Prototype**

```
int SingleAxisSlaveModeEnable(
    int SocketID,
    char * GroupName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
GroupName	char *	SingleAxis group name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -41: Slave-master mode not configured.

### 6.2.1.295 SingleAxisSlaveParametersGet

#### Name

**SingleAxisSlaveParametersGet** – Returns the slave parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a SingleAxis group): (-8)
- Check the positioner name: (-18)
- Check the slave parameters (must be configured): (-22)

#### Description

This function returns the slave parameters: the master positioner name and the master-slave ratio.

#### Prototype

```
int SingleAxisSlaveParametersGet(  
    int SocketID,  
    char * GroupName,  
    int NbPositioners,  
    char * MasterPositionerName ,  
    double * Ratio  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	SingleAxis group name.

#### Output parameters

MasterPositionerName	char *	Master positioner name from any group.
Ratio	double *	Gear ratio between the master and the slave.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.296 SingleAxisSlaveParametersSet**

#### **Name**

**SingleAxisSlaveParametersSet** – Sets the slave parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the master group type: (-8)
- Check the ratio value (Ratio >0): (-17)
- Check the master positioner name: (-18)
- Check the slave parameters (must be configured): (-22)
- Check the base velocity value (must be null): (-48)

#### **Description**

This function configures the slave parameters only for a SingleAxis group.

The slave is a copy of the master and a ratio can be applied: Slave = Ratio \* Master.

The slave-master mode is activated only after the call of “SingleAxisSlaveModeEnable” function.

The master can be a positioner from any group, except from a spindle group. If the master group is a spindle then (-22) error is returned. The master positioner must be different from the slave positioner else (-8) error is returned.

#### **NOTE**

**After an emergency stop, the master group and the slave group are in “NOTINIT” status. To restart a master-slave relation, the slave group(s) must be reinitialized before the master group.**

#### **Prototype**

```
int SingleAxisSlaveParametersSet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    char * MasterPositionerName ,
    double Ratio
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	SingleAxis group name.
MasterPositionerName	char *	Master positioner name from any group.
Ratio	double	Gear ratio between the master and the slave.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -48: BaseVelocity must be null.

### 6.2.1.297 SingleAxisThetaClampDisable

#### Name

**SingleAxisThetaClampDisable** – unclamps a SingleAxisTheta group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### Description

This function unclamps the selected SingleAxisTheta group

The group must be in the CLAMPED state. If unclamping is successful, the group is unclamped and the group state becomes “READY”.

#### Prototype

```
int SingleAxisThetaClampDisable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	SingleAxis group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.298 SingleAxisThetaClampEnable

#### Name

**SingleAxisThetaClampEnable** – clamps a SingleAxisTheta group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group): (-8)
- Valid positioner name: (-18)
- Valid group name: (-19)

#### Description

This function clamps the selected SingleAxisTheta group.

The group must be in the READY state. If clamping is successful, the group is clamped and the group state becomes “CLAMPED”.

#### Prototype

```
int SingleAxisThetaClampEnable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	SingleAxis group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.299 SingleAxisThetaFeedforwardJerkParametersGet**

#### **Name**

**SingleAxisThetaFeedforwardJerkParametersGet** – Gets currently used “XY to Theta Feedforward Jerk” feature parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group name: (-19)

#### **Description**

The “XY to Theta Feedforward Jerk” feature allows to compensate the vibration effect on the Theta stage when the XY stage speeds up or speeds down.

This function gets the currently used correction gains of “XY to Theta Feedforward Jerk” feature.

This API is reserved to SingleAxisTheta group only.

#### **Prototype**

```
int SingleAxisThetaFeedforwardJerkParametersGet(
    int SocketID,
    char GroupName[250],
    double * KFeedforwardJerkX,
    double * KFeedforwardJerkY
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the _ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

KFeedforwardJerkX	double *	X axis feedforward Jerk gain.
KFeedforwardJerkY	double *	Y axis feedforward Jerk gain.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -7: Wrong format in the command string.
- -8: Wrong object type for this command.
- -9: Wrong number of parameters in the command.
- -14: Wrong parameter type in the command string: double or double \* expected.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.300 SingleAxisThetaFeedforwardJerkParametersSet**

#### **Name**

**SingleAxisThetaFeedforwardJerkParametersSet** – Sets “XY to Theta Feedforward Jerk” feature parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group name: (-19)

#### **Description**

The “XY to Theta Feedforward Jerk” feature allows to compensate the vibration effect on the Theta stage when the XY stage speeds up or speeds down.

This function configures the correction gains of “XY to Theta Feedforward Jerk” feature for the SingleAxisTheta group.

This API is reserved to SingleAxisTheta group only.

#### **Prototype**

```
int SingleAxisThetaFeedforwardJerkParametersSet(
    int SocketID,
    char GroupName[250],
    double KFeedforwardJerkX,
    double KFeedforwardJerkY
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	SingleAxisTheta Group name.
KFeedforwardJerkX	double	X axis feedforward Jerk gain.
KFeedforwardJerkY	double	Y axis feedforward Jerk gain.

#### **Output parameters**

None.

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -7: Wrong format in the command string.
- -8: Wrong object type for this command.
- -9: Wrong number of parameters in the command.
- -14: Wrong parameter type in the command string: double or double \* expected.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.301 SingleAxisThetaFeedforwardParametersGet

#### Name

**SingleAxisThetaFeedforwardParametersGet** – Gets currently used “XY to Theta Feedforward” feature parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group name: (-19)

#### Description

The “XY to Theta Feedforward” feature allows to compensate the vibration effect on the Theta stage when the XY stage speeds up or speeds down.

This function gets the currently used correction gains of “XY to Theta Feedforward” feature.

This API is reserved to SingleAxisTheta group only.

#### Prototype

```
Int SingleAxisThetaFeedforwardParametersGet(
```

```
    int SocketID,  
    char PositionerName[250],  
    double * KFeedforwardX,  
    double * KFeedforwardY  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

KFeedforwardX	double *	X axis feedforward gain.
KFeedforwardY	double *	Y axis feedforward gain.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.302 SingleAxisThetaFeedforwardParametersSet**

#### **Name**

**SingleAxisThetaFeedforwardParametersSet** – Sets “XY to Theta Feedforward” feature parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group name: (-19)

#### **Description**

The “XY to Theta Feedforward” feature allows to compensate the vibration effect on the Theta stage when the XY stage speeds up or speeds down.

This function configures the correction gains of “XY to Theta Feedforward” feature for the SingleAxisTheta group.

This API is reserved to SingleAxisTheta group only.

#### **Prototype**

```
Int SingleAxisThetaFeedforwardParametersSet(
```

```
    int SocketID,  
    char * PositionerName,  
    double KFeedforwardX,  
    double KFeedforwardY  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
KFeedforwardX	double	X axis feedforward gain.
KFeedforwardY	double	Y axis feedforward gain.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.303 SingleAxisThetaSlaveModeDisable

#### Name

**SingleAxisThetaSlaveModeDisable** – Disables the slave-master mode.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the group type (must be a SingleAxisTheta group): (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group state must be "SLAVE": (-22)

#### Description

This function disables the master-slave mode. If a motion is in progress then it is aborted.

To use this function, the group state must be SLAVE (46). If it's not the case then (-22) is returned.

#### Prototype

```
int SingleAxisThetaSlaveModeDisable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
GroupName	char *	SingleAxisTheta group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.304 SingleAxisThetaSlaveModeEnable**

#### **Name**

**SingleAxisThetaSlaveModeEnable** – Enables the slave-master mode.

#### **Input tests**

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the group type (must be a SingleAxisTheta group): (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group state must be "READY": (-22)
- Check the slave parameters (must be configured): (-41)

#### **Description**

This function enables the master-slave mode only if the slave group is in "READY" state. In this mode the slave must be defined as a SingleAxisTheta group and the master can be a positioner from any group.

To use this function, the SingleAxisTheta group must be in the READY state. If it's not the case then (-22) error is returned.

To use this function, the master positioner and the slave ratio must be configured using the "SingleAxisThetaSlaveParametersSet" function. If it's not the case then (-41) error is returned.

#### **Prototype**

```
int SingleAxisThetaSlaveModeEnable(
    int SocketID,
    char * GroupName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
GroupName	char *	SingleAxisTheta group name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -41: Slave-master mode not configured.

### 6.2.1.305 SingleAxisThetaSlaveParametersGet

#### Name

**SingleAxisThetaSlaveParametersGet** – Returns the slave parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a SingleAxisTheta group): (-8)
- Check the positioner name: (-18)
- Check the slave parameters (must be configured): (-22)

#### Description

This function returns the slave parameters: the master positioner name and the master-slave ratio.

#### Prototype

```
int SingleAxisThetaSlaveParametersGet(  
    int SocketID,  
    char * GroupName,  
    int NbPositioners,  
    char * MasterPositionerName ,  
    double * Ratio  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	SingleAxisTheta group name.

#### Output parameters

MasterPositionerName	char *	Master positioner name from any group.
Ratio	double *	Gear ratio between the master and the slave.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.306 SingleAxisThetaSlaveParametersSet**

#### **Name**

**SingleAxisThetaSlaveParametersSet** – Sets the slave parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the master group type: (-8)
- Check the ratio value (Ratio >0): (-17)
- Check the master positioner name: (-18)
- Check the slave parameters (must be configured): (-22)
- Check the base velocity value (must be null): (-48)

#### **Description**

This function configures the slave parameters only for a SingleAxisTheta group.

The slave is a copy of the master and a ratio can be applied: Slave = Ratio \* Master.

The slave-master mode is activated only after the call of “SingleAxisThetaSlaveModeEnable” function.

The master can be a positioner from any group, except from a spindle group. If the master group is a spindle then (-22) error is returned. The master positioner must be different from the slave positioner else (-8) error is returned.

#### **NOTE**

**After an emergency stop, the master group and the slave group are in “NOTINIT” status. To restart a master-slave relation, the slave group(s) must be reinitialized before the master group.**

#### **Prototype**

```
int SingleAxisThetaSlaveParametersSet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    char * MasterPositionerName ,
    double Ratio
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	SingleAxisTheta group name.
MasterPositionerName	char *	Master positioner name from any group.
Ratio	double	Gear ratio between the master and the slave.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -48: BaseVelocity must be null.

### 6.2.1.307 SpindleSlaveModeDisable

#### Name

**SpindleSlaveModeDisable** – Disables the slave-master mode.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the group type (must be a Spindle group): (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group state must be "SLAVE": (-22)

#### Description

This function disables the master-slave mode for a spindle group. If a motion is in progress then it is aborted.

To use this function, the group state must be SLAVE (46). If it's not the case then (-22) error is returned.

#### Prototype

```
int SpindleSlaveModeDisable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
GroupName	char *	Spindle group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.308 SpindleSlaveModeEnable**

#### **Name**

**SpindleSlaveModeEnable** – Enables the slave-master mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a Spindle group): (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group state must be “READY”: (-22)
- Check the slave parameters (must be configured): (-41)

#### **Description**

This function enables the master-slave mode only if the slave group is in ready mode. In this mode the slave must be defined as a Spindle group and the master can be a positioner from any group.

To use this function, the Spindle group must be in the READY state. If it's not the case then (-22) error is returned.

To use this function, the master positioner and the slave ratio must be configured with the “SpindleSlaveParametersSet” function. If it's not the case then (-41) error is returned.

#### **Prototype**

```
int SpindleSlaveModeEnable(
    int SocketID,
    char * GroupName
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Spindle group name.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -41: Slave-master mode not configured.

### 6.2.1.309 SpindleSlaveParametersGet

#### Name

**SpindleSlaveParametersGet** – Returns the spindle slave parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a Spindle group): (-8)
- Check the positioner name: (-18)
- Check the slave parameters (must be configured): (-22)

#### Description

This function returns the slave parameters: the master positioner name and the master-slave ratio.

#### Prototype

```
int SpindleSlaveParametersGet(  
    int SocketID,  
    char * GroupName,  
    int NbPositioners,  
    char * MasterPositionerName ,  
    double * Ratio  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Spindle group name.

#### Output parameters

MasterPositionerName	char *	Master positioner name from any group.
Ratio	double *	Gear ratio between the master and the slave.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.310 SpindleSlaveParametersSet**

#### **Name**

**SpindleSlaveParametersSet** – Sets the spindle slave parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the master group type: (-8)
- Check the ratio value (Ratio >0): (-17)
- Check the master positioner name: (-18)
- Check the slave parameters (must be configured): (-22)
- Check the base velocity value (must be null): (-48)

#### **Description**

This function configures the slave parameters only for a Spindle group.

The slave is a master copy and a ratio can be applied: Slave = Ratio \* Master.

The slave-master mode is activated only after calling of “SingleAxisSlaveModeEnable” function.

The master can be a positioner from a spindle group only. If the master group is another group type then (-22) error is returned. The master positioner must be different from the slave positioner else (-8) error is returned.

#### **NOTE**

**After an emergency stop, the master group and the slave group are in “NOTINIT” state. To restart a master-slave relation, the slave group(s) must be reinitialized before the master group.**

#### **Prototype**

```
int SpindleSlaveParametersSet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    char * MasterPositionerName ,
    double * Ratio
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Spindle group name.
MasterPositionerName	char *	Master positioner name from any group.
Ratio	double	Gear ratio between the master and the slave.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -48: BaseVelocity must be null.

### **6.2.1.311 `TCLScriptExecute`**

#### **Name**

**TCLScriptExecute** – Executes a TCL script.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check TCL file name: (-36)
- Check TCL interpreter (task loading): (-37)
- Check task name: (-47)

#### **Description**

This function executes a TCL script. The TCL script file must be saved in the folder “..\Public\Scripts” of the XPS controller.

- *TaskName* is a user designation for the TCL script being executed. If two TCL scripts are executed at the same time with the same task name, The (-47) error is returned because having the same TaskName is not allowed.
- *InputArguments* represents the input arguments of the TCL script to be executed. The number of these input arguments is not limited but the string length is limited to 250 characters. The argument separator is a comma.

#### **Prototype**

```
int TCLScriptExecute(  
    int SocketID,  
    char * TCLFileName,  
    char * TaskName,  
    char * InputArguments  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
TCLFileName	char *	File name contains the TCL script.
TaskName	char *	Task name.
InputArguments	char *	Input argument string (separator is a comma).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -36: Unknown TCL file.
- -37: TCL interpreter doesn't run.
- -47: Wrong TCL task name: each TCL task name must be different.

### **6.2.1.312 `TCLScriptExecuteAndWait`**

#### **Name**

**TCLScriptExecuteAndWait** – Executes a TCL script and waits until the end of execution.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check TCL file name: (-36)
- Check TCL interpreter (task loading): (-37)
- Check task name: (-47)

#### **Description**

This function executes a TCL program. The “TCLScriptExecuteAndWait” function is different than the “TCLScriptExecute” function because it blocks the socket until the script terminates. The TCL script file must be saved in the folder “..\\Public\\Scripts” of the XPS controller. The file extension is “.tcl”.

- *TaskName* is a user designation for the TCL script in execution. If two TCL scripts are executed at the same time with the same task name, The (-47) error is returned because having the same TaskName is not allowed.
- *InputArguments* represents the input arguments of the TCL script to be executed. The number of these input arguments is not limited but the string length is limited to 250 characters. The argument separator is a comma.
- *OutputArguments* represents the output arguments of the TCL script to be executed. The number of these output arguments is not limited but the string length is limited to 250 characters. The argument separator is a comma.

#### **Prototype**

```
int TCLScriptExecuteAndWait(
    int SocketID,
    char * TCLFileName,
    char * TaskName,
    char * InputArguments,
    char * OutputArguments
    )
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
TCLFileName	char *	File name contains the TCL script.
TaskName	char *	Task name.
InputArguments	char *	Input argument string (separator is a comma).

#### **Output parameters**

OutputArguments	char *	Output argument string (separator is a comma).
-----------------	--------	--

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -36: Unknown TCL file.
- -37: TCL interpreter doesn't run.
- -47: Wrong TCL task name: each TCL task name must be different.

### **6.2.1.313 `TCLScriptExecuteWithPriority`**

#### **Name**

**TCLScriptExecuteWithPriority** – Executes a TCL script with TCL task given priority.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Priority mnemonic incorrect: (-17)
- Check TCL file name: (-36)
- Check TCL interpreter (task loading): (-37)
- Check task name: (-47)

#### **Description**

This function executes a TCL script with a TCL task and a user-defined priority level. The TCL script file must be saved in the folder “..\\Public\\Scripts” of the XPS controller.

- *TaskName* is a user designation for the TCL script in execution. If two TCL scripts are executed at the same time with the same task name, The (-47) error is returned because having the same TaskName is not allowed.
- *InputArguments* represents the input arguments to the TCL script to be executed. The number of these input arguments is not limited but the string length is limited to 250 characters. The argument separator is a comma.
- *PriorityLevel* has three possible values: “HIGH”, “MEDIUM” and “LOW”, with the order being HIGH > MEDIUM > LOW.

#### **Prototype**

```
int TCLScriptExecuteWithPriority(  
    int SocketID,  
    char * TCLFileName,  
    char * TaskName,  
    char * Priority,  
    char * InputArguments  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
TCLFileName	char *	File name contains the TCL script.
TaskName	char *	Task name.
Priority	char *	TCL task priority (HIGH, MEDIUM or LOW).
InputArguments	char *	Input argument string (separator is a comma).

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -36: Unknown TCL file.
- -37: TCL interpreter doesn't run.
- -47: Wrong TCL task name: each TCL task name must be different.

### 6.2.1.314 TCLScriptKill

#### Name

**TCLScriptKill** – Kills a TCL script.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check semaphore to use the TCL interpreter: (-37)
- Check TCL interpreter (task loading) and task name: (-38)

#### Description

This function kills a running TCL script selected using its task name. The task name is a user designation for the TCL script in execution.

---

#### NOTE

**For the boot script, the task name is “BootScript”.**

---

#### Prototype

```
int TCLScriptKill(  
    int SocketID,  
    char * TaskName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
TaskName	char *	Task name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -37: TCL interpreter doesn't run.
- -38: TCL script can not be killed: Wrong task name or task does not run.
- -47: Wrong TCL task name: each TCL task name must be different.

### 6.2.1.315 TCLScriptKillAll

#### Name

**TCLScriptKillAll** – Kills all running TCL scripts.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check TCL interpreter (task loading) and task name: (-38)

#### Description

This function kills all running TCL scripts.

#### Prototype

```
int TCLScriptKillAll(  
    int SocketID  
)  
    Input parameters  
    SocketID           int           Socket identifier gets by the  
                                "TCP_ConnectToServer" function.
```

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -38: TCL script can not be killed: Wrong task name or task does not run.

### 6.2.1.316 TCLScriptRunningListGet

#### Name

**TCLScriptRunningListGet**– Gets the list of all TCL processes in progress.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the task name’s list of all TCL scripts in progression.

#### Prototype

```
int TCLScriptRunningListGet(  
    int SocketID,  
    char * TCLScriptsList  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

TCLScriptsList	char *	List of TCL scripts in progression.
----------------	--------	-------------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.317 TCP\_CloseSocket

#### Name

**TCP\_CloseSocket** – Closes a socket.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check socket identifier (Max = 100).
- Socket must be used.

#### Description

Closed the opened TCP/IP communication defined by the given socket identifier. If the socket is undefined or is not used, then nothing happens.

#### Prototype

```
void TCP_CloseSocket(  
    int SocketID  
)
```

#### Input parameters

SocketID                    int                    Socket identifier used in each function.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

None.

### 6.2.1.318 TCP\_ConnectToServer

#### Name

**TCP\_ConnectToServer** – Configures TCP/IP communication and opens a socket.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check number of used sockets (Max = 100): if no free socket then the SocketID is set to -1.

#### Description

Configures the TCP/IP communication and opens a socket to connect TCP server.

This function returns a socket identifier to use for each function call. The socket identifier is defined between 0 to 99. If the TCP/IP connection failed then the “SocketID” value is -1.

---

#### NOTE

**OpenConnection** function is used when users are in local mode, it only needs the timeout and socket number to open the connection with the XPS controller.

**TCP\_ConnectToServer** function needs more information like the port number and the IP address. This function is called with the DLL.

---

#### Prototype

```
int TCP_ConnectToServer(  
    char * IP_Address,  
    int IP_Port,  
    double TimeOut  
)
```

#### Input parameters

IP_Address	char *	TCP IP address: 195.168.33.xxx or another.
IP_Port	int	TCP IP port: 5001 for XPS controller.
TimeOut	double	Timeout in seconds used for each function execution.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

SocketID                    int                    Socket identifier used in each function.

### 6.2.1.319 TCP\_GetError

#### Name

**TCP\_GetError** – Gets the last error about socket.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check socket identifier (Max = 100).
- Socket must be used.

#### Description

Gets the last error from the socket defined by the given socket identifier. If the socket is undefined or is not used, the error description is blank.

#### Prototype

```
int TCP_GetError(  
    int SocketID,  
    char * ErrorString  
)
```

#### Input parameters

SocketID                    int                 Socket identifier used in each function.

#### Output parameters

ErrorString                char \*              Last error description.

#### Return (In addition to the results of testing the inputs of functions)

None.

### 6.2.1.320 TCP\_SetTimeout

#### Name

**TCP\_SetTimeout** – Configures the timeout for TCP/IP communication.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check number of used sockets (Maximum number = 100).
- Socket must be used.
- Timeout value must be positive.

#### Description

Sets a new timeout value in seconds for the opened TCP/IP communication defined by a socket identifier.

If the timeout is less than 0.001, the timeout value defaults to 0.001.

If the socket is undefined or is not used then nothing happens.

#### Prototype

```
int TCP_SetTimeout(  
    int SocketID,  
    double Timout  
)
```

#### Input parameters

SocketID	int	Socket identifier used in each function.
Timeout	double	Timeout in seconds used for each function execution.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

None.

### 6.2.1.321 TimerGet

#### Name

**TimerGet** – Gets the number of frequency ticks for the selected timer.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the number of frequency ticks configured for the selected timer.

The “TimerName” can be defined as:

- Timer1
- Timer2
- Timer3
- Timer4
- Timer5

The “FrequencyTicks” defines the frequency of the timer:

One frequency tick represents a corrector period = >0.125 ms = >8 khz

N frequency ticks represents N corrector periods = >N \* 0.125 ms = > $\frac{8}{N}$  kHz

#### NOTE

“FrequencyTicks” = 0 means that the timer is disabled.

#### Prototype

```
int TimerGet(
    int SocketID,
    char * TimerName,
    int * FrequencyTicks
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
TimerName	char *	Name of timer.

#### Output parameters

FrequencyTicks	int *	Number of frequency ticks.
----------------	-------	----------------------------

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.322 TimerSet

#### Name

**TimerSet** – Sets the number of frequency ticks for the selected timer.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function sets the number of frequency ticks for the selected timer to activate it.

The “TimerName” can be defined as:

- Timer1
- Timer2
- Timer3
- Timer4
- Timer5

The “FrequencyTicks” allows to define the frequency of the timer:

One frequency tick represents a corrector period = >0.125 ms = >8 kHz

$$N \text{ frequency ticks represents } N \text{ corrector periods} = >N * 0.125 \text{ ms} = >\frac{8}{N} \text{ kHz}$$

#### NOTE

“FrequencyTicks” = 0 means that the timer is disabled.

#### Prototype

```
int TimerSet(
    int SocketID,
    char * TimerName,
    int FrequencyTicks
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
TimerName	char *	Name of timer.
FrequencyTicks	int	Number of frequency ticks.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.1.323 TZFocusModeDisable

#### Name

**TZFocusModeDisable** – Disables the TZ group from out of the FOCUS state.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the group type (must be a TZ group): (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group state must be "READY": (-22)

#### Description

This function disables the FOCUS mode of a TZ group. If it executes the group quits FOCUS state and goes into READY state.

To use this function, The group must be in a FOCUS state. If not then (-22) error is returned.

#### Prototype

```
int TZFocusModeDisable(  
    int SocketID,  
    char * GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	TZ group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.324 TZFocusModeEnable

#### Name

**TZFocusModeEnable** – Enables the TZ group to go in FOCUS state.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".
- Check the group type (must be a TZ group): (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group state must be "READY": (-22)

#### Description

This function enables the focus mode for the TZ group.

To use this function, The TZ group must be in READY state. If it's not then (-22) error is returned.

#### Prototype

```
int TZFocusModeEnable(  
    int SocketID,  
    char *GroupName  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
GroupName	char *	TZ group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.325 TZMotorDecouplingMatrixGet

#### Name

**TZMotorDecouplingMatrixGet** – Gets TZ motor decoupling matrix parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group name: (-19)

#### Description

This function gets the parameters of motor decoupling matrix for the TZ group.

TZ motor decoupling matrix:

```
[ Param1  Param2      Param3 ]  
[ Param4  Param5      Param6 ]  
[ Param7  Param8      Param9 ]
```

This API is reserved to TZ group only.

#### Prototype

```
int TZMotorDecouplingMatrixGet(  
    int SocketID,  
    char GroupName[250],  
    double * Param1,  
    double * Param2,  
    double * Param3,  
    double * Param4,  
    double * Param5,  
    double * Param6,  
    double * Param7,  
    double * Param8,  
    double * Param9  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

### **Output parameters**

Param1	double * Motor decoupling matrix coefficient.
Param2	double * Motor decoupling matrix coefficient.
Param3	double * Motor decoupling matrix coefficient.
Param4	double * Motor decoupling matrix coefficient.
Param5	double * Motor decoupling matrix coefficient.
Param6	double * Motor decoupling matrix coefficient.
Param7	double * Motor decoupling matrix coefficient.
Param8	double * Motor decoupling matrix coefficient.
Param9	double * Motor decoupling matrix coefficient.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.326 TZMotorDecouplingMatrixSet

#### Name

**TZMotorDecouplingMatrixSet** – Sets TZ motor decoupling matrix parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group name: (-19)

#### Description

This function configures the parameters of motor decoupling matrix for the TZ group.

The values are set in the decoupling matrix in the following order:

```
[ Param1  Param2      Param3 ]  
[ Param4  Param5      Param6 ]  
[ Param7  Param8      Param9 ]
```

This API is reserved to TZ group only.

#### Prototype

```
int TZMotorDecouplingMatrixSet(  
    int SocketID,  
    char GroupName[250],  
    double Param1,  
    double Param2,  
    double Param3,  
    double Param4,  
    double Param5,  
    double Param6,  
    double Param7,  
    double Param8,  
    double Param9  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
Param1	double	Motor decoupling matrix coefficient.
Param2	double	Motor decoupling matrix coefficient.
Param3	double	Motor decoupling matrix coefficient.
Param4	double	Motor decoupling matrix coefficient.
Param5	double	Motor decoupling matrix coefficient.
Param6	double	Motor decoupling matrix coefficient.
Param7	double	Motor decoupling matrix coefficient.
Param8	double	Motor decoupling matrix coefficient.
Param9	double	Motor decoupling matrix coefficient.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.327 TZPTExecution**

#### **Name**

**TZPTExecution** – Executes a PT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length: (-3)
- Check group type (must be a TZ group): (-8)
- Check input value (number of executions must >0): (-17)
- Check group name: (-19)
- Group state must be "READY": (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Check backlash (must not be enabled): (-46)
- Check BaseVelocity (stages.ini, must = 0): (-48)
- Check trajectory file existence or file reading: (-61)
- Check message queue: (-71)

#### **Description**

This function executes a PT (Position Time) trajectory. The trajectory file must be stored in the folder “\Admin\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (message queue or task error) then (-72) is returned.

Before a trajectory execution, it is recommended to check whether the trajectory is within the positioner motion capabilities by using “TZPTVerification” and “TZPTVerificationResultGet” functions.

During the trajectory execution, if a positioner reaches one of travel limits, the trajectory execution will stop and the (-25) error is generated in the positioner errors.

---

#### **NOTE**

**In case of an (-33) error, an (-25) error or (-44) error, the group state becomes DISABLE. To help determine the error source, check the positioner errors, the hardware status and the driver status.**

**This function can be used only with the XPS-Q Precision Platform controller.**

---

#### **Prototype**

```
int TZPTExecution(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250],  
    int ExecutionNumber  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
ExecutionNumber	int	Number of trajectory executions.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.
- -25: Following Error.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -46: Not allowed action due to backlash.
- -48: BaseVelocity must be null.
- -61: Error file corrupt or file doesn't exist.
- -71: Error read from or write in message queue.
- -72: Error trajectory initialization.

### 6.2.1.328 **TZPTLoadToMemory**

#### Name

**TZPTLoadToMemory** – Loads some lines of PT trajectory to the controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function loads some lines of PT trajectory into XPS controller memory. Each trajectory element must be separated by a comma. The trajectory lines are separated between them by a “\n” (LF) character. To verify or to execute the PT trajectory loaded in memory, use the string “**FromMemory**” instead of a file name.

#### NOTE

All of the PT functions, when called with the string “**FromMemory**” instead of a **FileName**, will perform the same operation as the PT trajectory in RAM as it does from a disk.

#### Example:

```
TZPTLoadToMemory(socketId,myGroup,"dT1,dX11,dX12,dX13|n...dTn,dXn1,dXn2,dXn3|n")
TZPTVerification (socketId,myGroup,FromMemory)
TZPTExecution(socketId,myGroup,FromMemory,I).
```

#### Prototype

```
int TZPTLoadToMemory(  
    int SocketID,  
    char GroupName[250],  
    char TrajectoryData[400]  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
TrajectoryData	char *	Trajectory data lines.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.329 **TZPTParametersGet**

#### **Name**

**TZPTParametersGet** – Gets PT trajectory parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Check current executing trajectory type (must be PT): (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function returns the PT trajectory parameters (trajectory name and current executing element number) of the current PT trajectory.

#### **Prototype**

```
int TZPTParametersGet(
    int SocketID,
    char GroupName[250],
    char *FileName,
    int *CurrentElementNumber
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

FileName	char *	Currently executing trajectory file name.
CurrentElementNumber	int *	Currently executing element number.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### **6.2.1.330 TZPTPulseOutputGet**

#### **Name**

**TZPTPulseOutputGet** – Gets the configuration of pulse generation of a PT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function returns the last configuration of pulse generation of a PT trajectory, that was previously set by *TZPTPulseOutputSet()*.

The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

#### ***Example:***

*TZPTPulseOutputSet(MyGroup, 3, 5, 0.01)*

*TZPTPulseOutputGet(MyGroup) => 0,3,5,0.01 (0 is error return, means OK)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

Start element= 3

End element = 5

Time interval = 0.01 seconds.

#### **Prototype**

int **TZPTPulseOutputGet(**

```
    int SocketID,
    char GroupName[250],
    int * StartElement,
    int * EndElement,
    double * TimeInterval
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

StartElement	int *	Start pulse element number.
EndElement	int *	End pulse element number.
TimeInterval	double *	Time interval between pulses (seconds).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.331 **TZPTPulseOutputSet**

#### Name

**TZPTPulseOutputSet** – Sets the configuration of pulse generation of a PT trajectory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Check the pulse generation must not be in progress: (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function configures and activates the pulse generation of a PT trajectory. The pulse generation is defined by a start element, an end element, and a time interval in seconds. If a pulse generation is already activated on the selected PT trajectory then this function returns -22 (“Not allowed action”) error.

Please note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, it is required to define the pulse output settings again.

This capability allows output of pulses at constant time intervals on a PT trajectory. The pulses are generated between the first and the last trajectory element. The minimum possible time interval is CorrectorISRPeriod value (*system.ref*).

The trajectory pulses are generated on the following GPIO outputs:

GPIO signals	ISA XPS controller	PCI XPS controller (for example XPS-RL) with basic GPIO board	PCI XPS controller (for example XPS-RL) with extended GPIO board
Window	GPIO2, pin 11	GPIO1.DO6	GPIO5.DO14
Pulses	GPIO2, pin 12	GPIO1.DO7	GPIO5.DO15

#### Example:

*TZPTPulseOutputSet(Group1, 3, 5, 0.01)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

#### Prototype

```
int TZPTPulseOutputSet(
    int SocketID,
    char GroupName[250],
    int StartElement,
    int EndElement,
    double TimeInterval
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
StartElement	int	Start pulse element number.
EndElement	int	End pulse element number.
TimeInterval	double	Time interval between pulses (seconds).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### 6.2.1.332 TZPTResetInMemory

#### Name

**TZPTResetInMemory** – Deletes the content of the PT trajectory buffer in the controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function deletes the PT trajectory buffer in the controller memory, that was previously loaded with the “TZPTLoadToMemory” function.

#### Prototype

```
int TZPTResetInMemory(  
    int SocketID,  
    char GroupName[250]  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.333 TZPTVerification

#### Name

**TZPTVerification** – Checks a PT trajectory data file.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length (must  $\leq 250$ ): (-3)
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Check BaseVelocity value (must = 0): (-48)
- Check trajectory file existence and the file format: (-61)
- Check trajectory (number of elements must  $> 0$ ): (-66)
- Check velocity (Minimum Velocity  $\leq$  Velocity  $\leq$  Maximum Velocity): (-68)
- Check acceleration (Minimum acc.  $\leq$  acceleration  $\leq$  Maximum acc.): (-69)
- Check end output velocity (must = 0): (-70)
- Check delta time (DeltaTime must  $> 0$ ): (-75)

#### Description

This function verifies the execution of a PT trajectory. The results of the verification can be got with the “TZPTVerificationResultGet” function. The trajectory file must be stored in the folder “\ADMIN\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (task error) then the (-72) error is returned.

This function can be executed at any time and is independent of the trajectory execution. It performs the following:

- Checks the trajectory file for data coherence.
- Calculates the trajectory limits, which are: the required travel per positioner, the maximum possible trajectory velocity and the maximum possible trajectory acceleration. This function helps define the parameters for the trajectory execution.
- The required travel values (MinimumPosition and MaximumPosition) are calculated relative to the position zero, not to the current position. So before executing a PT trajectory, the user must pay attention to the current position of the positioners to make sure that the trajectory will not exceed the positioner travel limits.
- If all is OK, it returns “SUCCESS” (0). Otherwise, it returns a corresponding error.

---

#### NOTE

**Because of the PT trajectory algorythm for elements end velocity calculation, a correct PT trajectory file must have at least two lines with zero displacements at the trajectory end. Otherwise, the “TZPTVerification” function returns the (-70) error.**

**The “TZPTVerification” function is independent from the “TZPTExecution” function, but it is highly recommended to execute this function before executing a PT trajectory.**

---

### **Prototype**

```
int TZPTVerification(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250]  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.
- -61: Error file corrupt or file doesn't exist.
- -66: Trajectory doesn't content any element.
- -68: Acceleration on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -70: Final velocity on trajectory is not zero.
- -72: Error trajectory initialization.
- -75: Trajectory element has a negative or null delta T.

### 6.2.1.334 TZPTVerificationResultGet

#### Name

**TZPTVerificationResultGet** – Get the results of the “TZPTVerification” function.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name length (must  $\leq 250$ ): (-3)
- Check positioner name: (-18)
- Check the last TZ PTVerification (must be done): (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function returns the results of the previous “TZPTVerification” function, positioner by positioner. The results are the travel requirements (min and max values), the possible maximum velocity and the possible maximum acceleration.

If no verification was previously done then the (-22) error is returned.

#### Prototype

```
int TZPTVerificationResultGet(  
    int SocketID,  
    char PositionerName[250],  
    char * TrajectoryFileName,  
    double * MinimumPosition,  
    double * MaximumPosition,  
    double * MaximumVelocity,  
    double * MaximumAcceleration  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

TrajectoryFileName	char *	Examined trajectory file name.
MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
MaximumVelocity	double *	Maximum velocity (units/s).
MaximumAcceleration	double *	Maximum acceleration (units/s <sup>2</sup> ).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### 6.2.1.335 TZPVTExecution

#### Name

**TZPVTExecution** – Execute a PVT trajectory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length: (-3)
- Check group type (must be a TZ group): (-8)
- Check input value (number of executions must >0): (-17)
- Check group name: (-19)
- Group state must be "READY": (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Check backlash (must not be enabled): (-46)
- Check BaseVelocity (stages.ini, must = 0): (-48)
- Check trajectory file existence or file reading: (-61)
- Check message queue: (-71)

#### Description

This function executes a PVT (Position Velocity Time) trajectory. The trajectory file must be stored in the folder “\Admin\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (message queue or task error) then (-72) is returned.

Before a trajectory execution, it is recommended to check whether the trajectory is within the positioner motion capabilities by using “TZPVTVerification” and “TZPVTVerificationResultGet” functions.

During the trajectory execution, if a positioner reaches one of travel limits, the trajectory execution will stop and the (-25) error is generated in the positioner errors.

---

#### NOTES

**In case of an (-33) error, an (-25) error or (-44) error, the group state becomes DISABLE. To help determine the error source, check the positioner errors, the hardware status and the driver status.**

---

#### Prototype

```
int TZPVTExecution(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250],  
    int ExecutionNumber  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
ExecutionNumber	int	Number of trajectory executions.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.
- -25: Following Error.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -46: Not allowed action due to backlash.
- -48: BaseVelocity must be null.
- -61: Error file corrupt or file doesn't exist.
- -71: Error read from or write in message queue.
- -72: Error trajectory initialization.

### 6.2.1.336 **TZPVTLoadToMemory**

#### **Name**

**TZPVTLoadToMemory** – Load some lines of PVT trajectory to the controller memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function loads some lines of PVT trajectory into XPS controller memory. Each trajectory element must be separated by a comma. The trajectory lines are separated between them by a “\n” (LF) character. To verify or to execute the PVT trajectory loaded in memory, use the string “**FromMemory**” instead of a file name.

#### **NOTES**

All of the PVT functions, when called with the string “FromMemory” instead of a **FileName**, will perform the same operation as the PVT trajectory in RAM as it does from a disk.

#### **Example:**

```
TZPVTLoadToMemory(socketId,myGroup,"dT1,dX11,Vout11,dX12,Vout12,
dX13,Vout13|n...dTn,dXn1,Voutn1,dXn2,Voutn2, dXn3,Voutn3|n")
TZPVTVerification (socketId,myGroup,FromMemory)
TZPVTEExecution(socketId,myGroup,FromMemory,I).
```

#### **Prototype**

```
int TZPVTLoadToMemory(
    int SocketID,
    char GroupName[250],
    char FileName[250],
    char TrajectoryData[400]
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
TrajectoryData	char *	Trajectory data lines.

#### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.337 TZPVTParametersGet

#### Name

**TZPVTParametersGet** – Gets PVT trajectory parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Check current executing trajectory type (must be PVT): (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function returns the PVT trajectory parameters (trajectory name and current executing element number) of the current PVT trajectory.

#### Prototype

```
int TZPVTParametersGet(  
    int SocketID,  
    char GroupName[250],  
    char * FileName,  
    int * CurrentElementNumber  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

FileName	char *	Currently executing trajectory file name.
CurrentElementNumber	int *	Currently executing element number.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### **6.2.1.338 TZPVT\_PulseOutputGet**

#### **Name**

**TZPVT\_PulseOutputGet** – Gets the configuration of pulse generation of a PVT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function returns the last configuration of pulse generation of a PVT trajectory, that was previously set by *TZPVT\_PulseOutputSet()*.

The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

#### ***Example:***

*TZPVT\_PulseOutputSet(MyGroup, 3, 5, 0.01)*

*TZPVT\_PulseOutputGet(MyGroup) => 0,3,5,0.01 (0 is error return, means OK)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

Start element= 3

End element = 5

Time interval = 0.01 seconds.

#### **Prototype**

```
int TZPVT_PulseOutputGet(
    int SocketID,
    char GroupName[250],
    int * StartElement,
    int * EndElement,
    double * TimeInterval
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

StartElement	int *	Start pulse element number.
EndElement	int *	End pulse element number.
TimeInterval	double *	Time interval between pulses (seconds).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### **6.2.1.339 TZPVT\_PulseOutputSet**

#### **Name**

**TZPVT\_PulseOutputSet** – Set the configuration of pulse generation of a PVT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Check the pulse generation must not be in progress: (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function configures and activates the pulse generation of a PVT trajectory. The pulse generation is defined by a start element, an end element, and a time interval in seconds. If a pulse generation is already activated on the selected PVT trajectory then this function returns -22 (“Not allowed action”) error.

Please note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, it is required to define the pulse output settings again.

This capability allows output of pulses at constant time intervals on a PVT trajectory. The pulses are generated between the first and the last trajectory element. The minimum possible time interval is CorrectorISRPeriod value (*system.ref*).

The trajectory pulses are generated on the following GPIO outputs:

GPIO signals	ISA XPS controller	PCI XPS controller (for example XPS-RL) with basic GPIO board	PCI XPS controller (for example XPS-RL) with extended GPIO board
Window	GPIO2, pin 11	GPIO1.DO6	GPIO5.DO14
Pulses	GPIO2, pin 12	GPIO1.DO7	GPIO5.DO15

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual, Appendix / General I/O Description.

#### ***Example:***

*TZPVT\_PulseOutputSet(Group1, 3, 5, 0.01)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

#### **Prototype**

```
int TZPVT_PulseOutputSet(
    int SocketID,
    char GroupName[250],
    int StartElement,
    int EndElement,
    double TimeInterval
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
StartElement	int	Start pulse element number.
EndElement	int	End pulse element number.
TimeInterval	double	Time interval between pulses (seconds).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### 6.2.1.340 TZPVTResetInMemory

#### Name

**TZPVTResetInMemory** – Deletes the content of the PVT trajectory buffer in the controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function deletes the PVT trajectory buffer in the controller memory, that was previously loaded with the “TZPVTLoadToMemory” function.

#### Prototype

```
int TZPVTLoadToMemory(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250],  
    char TrajectoryData[400]  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
TrajectoryData	char *	Trajectory data lines.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.341 TZPVTVerification

#### Name

**TZPVTVerification** – Check a PVT trajectory data file.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length (must  $\leq 250$ ): (-3)
- Check group type (must be a TZ group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Check BaseVelocity value (must = 0): (-48)
- Check trajectory file existence and the file format: (-61)
- Check trajectory (number of elements must  $> 0$ ): (-66)
- Check velocity (Minimum Velocity  $\leq$  Velocity  $\leq$  Maximum Velocity): (-68)
- Check acceleration (Minimum acc.  $\leq$  acceleration  $\leq$  Maximum acc.): (-69)
- Check end output velocity (must = 0): (-70)
- Check delta time (DeltaTime must  $> 0$ ): (-75)

#### Description

This function verifies the execution of a PVT trajectory. The results of the verification can be got with the “TZPVTVerificationResultGet” function. The trajectory file must be stored in the folder “\ADMIN\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (task error) then the (-72) error is returned.

This function can be executed at any time and is independent of the trajectory execution. It performs the following:

- Checks the trajectory file for data coherence.
- Calculates the trajectory limits, which are: the required travel per positioner, the maximum possible trajectory velocity and the maximum possible trajectory acceleration. This function helps define the parameters for the trajectory execution.
- The required travel values (MinimumPosition and MaximumPosition) are calculated relative to the position zero, not to the current position. So before executing a PVT trajectory, the user must pay attention to the current position of the positioners to make sure that the trajectory will not exceed the positioner travel limits.
- If all is OK, it returns “SUCCESS” (0). Otherwise, it returns a corresponding error.

---

#### NOTES

The “TZPVTVerification” function is independent from the “TZPVTExecution” function, but it is highly recommended to execute this function before executing a PVT trajectory.

**This function can be used only with the XPS-Q Precision Platform controller.**

---

### **Prototype**

```
int TZPVTVerification(
    int SocketID,
    char GroupName[250],
    char FileName[250]
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.
- -61: Error file corrupt or file doesn't exist.
- -66: Trajectory doesn't content any element.
- -68: Acceleration on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -70: Final velocity on trajectory is not zero.
- -72: Error trajectory initialization.
- -75: Trajectory element has a negative or null delta T.

### **6.2.1.342 TZPVTVerificationResultGet**

#### **Name**

**TZPVTVerificationResultGet** – Get the results of the “TZPVTVerification” function.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name length (must  $\leq 250$ ): (-3)
- Check positioner name: (-18)
- Check the last TZ PVTVerification (must be done): (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function returns the results of the previous “TZPVTVerification” function, positioner by positioner. The results are the travel requirements (min and max values), the possible maximum velocity and the possible maximum acceleration.

If no verification was previously done then the (-22) error is returned.

#### **Prototype**

```
int TZPVTVerificationResultGet(
    char PositionerName[250],
    char * TrajectoryFileName,
    double * MinimumPosition,
    double * MaximumPosition,
    double * MaximumVelocity,
    double * MaximumAcceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

TrajectoryFileName	char *	Examined trajectory file name.
MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
MaximumVelocity	double *	Maximum velocity (units/s).
MaximumAcceleration	double *	Maximum acceleration (units/s <sup>2</sup> ).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### **6.2.1.343 TZTrackingCutOffFrequencyGet**

#### **Name**

**TZTrackingCutOffFrequencyGet** – Get tracking cut-off frequency of TZ group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a TZ group): (-8)
- Check the group (must not be a positioner): (-19)

#### **Description**

This function gets the tracking cut-off frequency of each positioner of TZ group.

#### **Prototype**

```
int TZTrackingCutOffFrequencyGet(
    int SocketID,
    char * GroupName,
    double * CutOffFrequency1,
    double * CutOffFrequency2,
    double * CutOffFrequency3
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function
GroupName	char *	TZ group name

#### **Output parameters**

CutOffFrequency1	double *	Positioner 1 cut-off frequency (Hz)
CutOffFrequency2	double *	Positioner 2 cut-off frequency (Hz)
CutOffFrequency3	double *	Positioner 3 cut-off frequency (Hz)

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.

### 6.2.1.344 TZTrackingCutOffFrequencySet

#### Name

**TZTrackingCutOffFrequencySet** – Set tracking cut-off frequency of TZ group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a TZ group): (-8)
- Value is out of range: (-17)
- Check the group (must not be a positioner): (-19)

#### Description

This function sets the tracking cut-off frequency for each positioner of TZ group.

#### Prototype

```
int TZTrackingCutOffFrequencySet(  
    int SocketID,  
    char * GroupName,  
    double CutOffFrequency1,  
    double CutOffFrequency2,  
    double CutOffFrequency3  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function
GroupName	char *	TZ group name
CutOffFrequency1	double	Positioner 1 cut-off frequency (Hz)
CutOffFrequency2	double	Positioner 2 cut-off frequency (Hz)
CutOffFrequency3	double	Positioner 3 cut-off frequency (Hz)

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.

### **6.2.1.345 TZTrackingUserMaximumZZZTargetDifferenceGet**

#### **Name**

**TZTrackingUserMaximumZZZTargetDifferenceGet** – Get tracking maximum ZZZ target difference of TZ group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a TZ group): (-8)
- Check the group (must not be a positioner): (-19)

#### **Description**

This function gets the tracking maximum ZZZ target difference of TZ group.

#### **Prototype**

```
int TZTrackingUserMaximumZZZTargetDifferenceGet(
    int SocketID,
    char * GroupName,
    double * UserMaximumZZZTargetDifference
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	TZ group name.

#### **Output parameters**

UserMaximumZZZTargetDifference double \* User maximum ZZZ target difference (units).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.

### **6.2.1.346 TZTrackingUserMaximumZZZTargetDifferenceSet**

#### **Name**

**TZTrackingUserMaximumZZZTargetDifferenceSet** – Sets tracking maximum ZZZ target difference for a TZ group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the group type (must be a TZ group): (-8)
- Value is out of range: (-17)
- Check the group (must not be a positioner): (-19)

#### **Description**

This function sets the tracking maximum ZZZ target difference for a TZ group.

#### **Prototype**

```
int TZTrackingUserMaximumZZZTargetDifferenceSet(  
    int SocketID,  
    char * GroupName,  
    double * UserMaximumZZZTargetDifference  
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	TZ group name.
UserMaximumZZZTargetDifference	double	User maximum ZZZ target difference (units).

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.347 XYGroupPositionCorrectedProfilerGet

#### Name

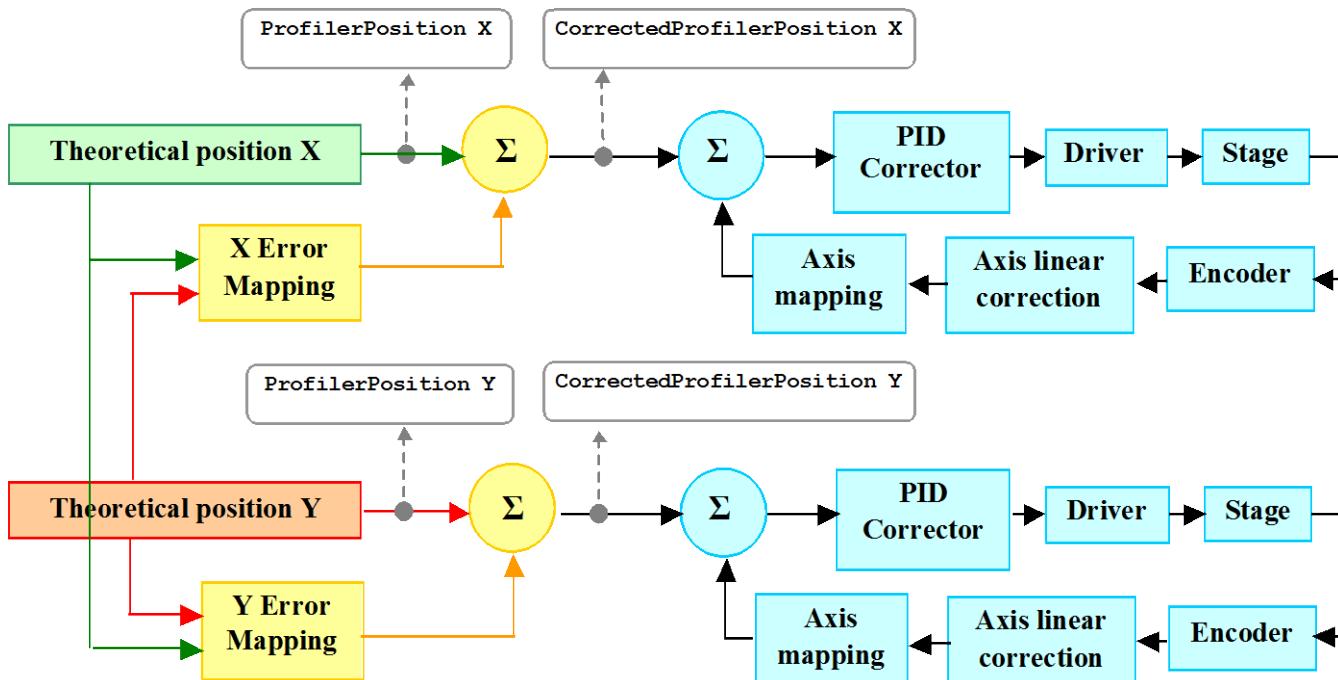
**XYGroupPositionCorrectedProfilerGet** – Returns the corrected profiler position for all positioners of an XY group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Valid group type (must be an XY group): (-18)
- Valid group name: (-19)

#### Description

This function gets the corrected position which is the theoretical position recalculated with the XY mapping correction.



This function applies the XY mapping on the theoretical user-defined positions and returns the corrected positions.

#### Prototype

```
int XYGroupPositionCorrectedProfilerGet(
    int SocketID,
    char * GroupName,
    double PositionX,
    double PositionY,
    double * CorrectedPositionX,
    double * CorrectedPositionY
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
PositionX	double	Theoretical position X.
PositionY	double	Theoretical position Y.

**Output parameters**

CorrectedPositionX	double *	Corrected theoretical position X.
CorrectedPositionY	double *	Corrected theoretical position Y.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.348 XYGroupPositionPCORawEncoderGet**

#### **Name**

**XYGroupPositionPCORawEncoderGet** – Returns the PCO raw encoder positions for an XY group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Valid group type (must be an XY group): (-18)
- Valid group name: (-19)

#### **Description**

This function returns the PCO raw encoder positions X and Y from the user specified positions X and Y.

#### **Prototype**

```
int XYGroupPositionPCORawEncoderGet(
    int SocketID,
    char * GroupName,
    double PositionX,
    double PositionY,
    double * PCORawPositionX,
    double * PCORawPositionY
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
PositionX	double	User corrected position X.
PositionY	double	User corrected position Y.

#### **Output parameters**

PCORawPositionX	double *	PCO Raw position X.
PCORawPositionY	double *	PCO Raw position Y.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.349 XYLineArcExecution

#### Name

**XYLineArcExecution** – Execute a LineArc trajectory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length: (-3)
- Check group type (must be a XY group): (-8)
- Check input value (number of executions must >0): (-17)
- Check input value (Velocity and Acceleration >0): (-17)
- Check group name: (-19)
- Group state must be "READY": (-22)
- Check backlash (must not be enabled): (-46)
- Check BaseVelocity (stages.ini, must = 0): (-48)
- Check trajectory file existence or file reading: (-61)
- Check the velocity (Velocity must ≤ TrajectoryMaximumVelocity): (-68)
- Check the acceleration (Acceleration must ≤ TrajectoryMaximumAcceleration): (-69)
- Check message queue: (-71)

#### Description

This function executes a LineArc trajectory. The trajectory file must be stored in the folder “\Admin\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (message queue or task error) then (-72) is returned.

Before a trajectory execution, it is recommended to check whether the trajectory is within the positioner motion capabilities by using “XYLineArcVerification” and “XYLineArcVerificationResultGet” functions.

During the trajectory execution, if a positioner reaches one of travel limits, the trajectory execution will stop and the (-25) error is generated in the positioner errors.

---

#### NOTE

**In case of an (-33) error, an (-25) error or (-44) error, the group state becomes DISABLE. To help determine the error source, check the positioner errors, the hardware status and the driver status.**

---

#### Prototype

```
int XYLineArcExecution(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250],  
    double Velocity,  
    double Acceleration,  
    int ExecutionNumber  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
Velocity	double	Trajectory velocity (units/s).
Acceleration	double	Trajectory acceleration (units/s <sup>2</sup> ).
ExecutionNumber	int	Number of trajectory executions.

**Output parameters**

None.

**Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -46: Not allowed action due to backlash.
- -48: BaseVelocity must be null.
- -61: Error file corrupt or file doesn't exist.
- -68: Velocity on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -71: Error read from or write in message queue.
- -72: Error trajectory initialization.

### **6.2.1.350 XYLineArcParametersGet**

#### **Name**

**XYLineArcParametersGet** – Get LineArc trajectory parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Check current executing trajectory type (must be LineArc): (-22)

#### **Description**

This function returns the LineArc trajectory parameters (trajectory name, trajectory velocity, trajectory acceleration, current executing element number) of the current LineArc trajectory.

#### **Prototype**

```
int XYLineArcParametersGet(
    int SocketID,
    char GroupName[250],
    char * FileName,
    double * Velocity,
    double * Acceleration,
    int * CurrentElementNumber
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

FileName	char *	Currently executing trajectory file name.
Velocity	double *	Trajectory velocity (units/s).
Acceleration	double *	Trajectory acceleration (units/s <sup>2</sup> ).
CurrentElementNumber	int *	Currently executing element number.

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.351 XYLineArcPulseOutputGet**

#### **Name**

**XYLineArcPulseOutputGet** – Get the configuration of pulse generation of a LineArc.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)

#### **Description**

This function returns the last configuration of pulse generation of a LineArc trajectory, that was previously set by *XYLineArcPulseOutputSet()*.

The pulse output configuration is defined by a pulse start trajectory curved length, a pulse end trajectory curved length, and a pulse trajectory curved length interval.

#### ***Example:***

*XYLineArcPulseOutputSet(MyGroup, 10, 30, 0.01)*

*XYLineArcPulseOutputGet(MyGroup) => 0,10,30,0.01 (0 is error return, means OK)*

One pulse will be generated every 10 µm on the LineArc trajectory between 10 mm and 30 mm trajectory curved lengths.

- Pulse start trajectory curved length = 10 mm
- Pulse end trajectory curved length = 30 mm
- Pulse trajectory curved length interval = 0.01 mm.

#### **Prototype**

```
int XYLineArcPulseOutputGet(
    int SocketID,
    char GroupName[250],
    double * StartLength,
    double * EndLength,
    double * PathLengthInterval
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

StartLength	double *	Pulse start length (units).
EndLength	double *	Pulse end length (units).
PathLengthInterval	double *	Pulse length interval (units).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.

### **6.2.1.352 XYLineArcPulseOutputSet**

#### **Name**

**XYLineArcPulseOutputSet** – Set the configuration of pulse generation of a LineArc trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Controller initialization failed: (-20)
- Check the pulse generation must not be in progress: (-22)

#### **Description**

This function configures and activates the pulse generation of a LineArc trajectory. The pulse generation is defined by a start element, an end element, and a time interval in seconds. If a pulse generation is already activated on the selected LineArc trajectory then this function returns -22 (“Not allowed action”) error.

Please note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, it is required to define the pulse output settings again.

This capability allows to generate pulses at constant pulse trajectory curved length intervals on a LineArc trajectory. The pulses are generated between a pulse start trajectory curved length and a pulse end trajectory curved length. All lengths are calculated in an orthogonal XY coordination system.

The trajectory pulses are generated on the following GPIO outputs:

GPIO signals	ISA XPS controller	PCI XPS controller (for example XPS-RL) with basic GPIO board	PCI XPS controller (for example XPS-RL) with extended GPIO board
Window	GPIO2, pin 11	GPIO1.DO6	GPIO5.DO14
Pulses	GPIO2, pin 12	GPIO1.DO7	GPIO5.DO15

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual, Appendix / General I/O Description.

#### ***Example:***

*XYLineArcPulseOutputSet(Group1, 10, 30, 0.01)*

One pulse will be generated every 10 µm on the LineArc trajectory between 10 mm and 30 mm trajectory curved lengths.

Pulse start trajectory curved length = 10 mm

Pulse end trajectory curved length = 30 mm

Pulse trajectory curved length interval = 0.01 mm

**Prototype**

```
int XYLineArcPulseOutputSet(  
    int SocketID,  
    char GroupName[250],  
    double StartLength,  
    double EndLength,  
    double PathLengthInterval  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
StartLength	double	Pulse start length (units).
EndLength	double	Pulse end length (units).
PathLengthInterval	double	Pulse length interval (units).

**Output parameters**

None.

**Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.353 XYLineArcVerification

#### Name

**XYLineArcVerification** – Check a LineArc trajectory data file.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length (must  $\leq 250$ ): (-3)
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Check trajectory file existence and the file format: (-61)
- Check trajectory element ( $Radius \geq 1e-14$ ): (-63)
- Check trajectory element ( $SweepAngle \geq 1e-14$ ): (-64)
- Check trajectory element ( $|XElementDistance| \geq 1e-14, |YElementDistance| \geq 1e-14, TangentOut \neq 1.797e308$ ): (-65)
- Check trajectory (number of elements must  $> 0$ ): (-66)
- Check keys (“*FirstTangent*” and “*DiscontinuityAngle*”) in trajectory file: (-74)

#### Description

This function verifies the execution of a LineArc trajectory. The results of the verification can be got with the “*XYLineArcVerificationResultGet*” function. The trajectory file must be stored in the folder “\ADMIN\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (task error) then the (-72) error is returned.

This function can be executed at any time and is independent of the trajectory execution. It performs the following:

- Checks the trajectory file for data coherence.
- Calculates the trajectory limits, which are: the required travel per positioner, the maximum possible trajectory velocity and the maximum possible trajectory acceleration. This function helps define the parameters for the trajectory execution.
- The required travel values (MinimumPosition and MaximumPosition) are calculated relative to the position zero, not to the current position. So before executing a LineArc trajectory, the user must pay attention to the current position of the positioners to make sure that the trajectory will not exceed the positioner travel limits.
- If all is OK, it returns “*SUCCESS*” (0). Otherwise, it returns a corresponding error.

---

#### NOTE

The “*XYLineArcVerification*” function is independent from the “*XYLineArcExecution*” function, but it is highly recommended to execute this function before executing a LineArc trajectory.

---

**Prototype**

```
int XYLineArcVerification(
    int SocketID,
    char GroupName[250],
    char FileName[250]
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.

**Output parameters**

None.

**Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -61: Error file corrupt or file doesn't exist.
- -63: Wrong XY trajectory element arc radius.
- -64: Wrong XY trajectory element sweep angle.
- -65: Trajectory line element discontinuity error or new element is too small.
- -66: Trajectory doesn't contain any element.
- -72: Error trajectory initialization.
- -74: Error file parameter key not found.

### **6.2.1.354 XYLineArcVerificationResultGet**

#### **Name**

**XYLineArcVerificationResultGet** – Get the results of the “XYLineArcVerification” function.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name length (must ≤250): (-3)
- Check positioner name: (-18)
- Check the last XY LineArcVerification (must be done): (-22)

#### **Description**

This function returns the results of the previous “XYLineArcVerification” function, positioner by positioner. The results are the travel requirements (min and max values), the possible maximum velocity and the possible maximum acceleration.

If no verification was previously done then the (-22) error is returned.

#### **Prototype**

```
int XYLineArcVerificationResultGet(
```

```
    SocketID,
    char PositionerName[250],
    char * TrajectoryFileName,
    double * MinimumPosition,
    double * MaximumPosition,
    double * MaximumVelocity,
    double * MaximumAcceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

TrajectoryFileName	char *	Examined trajectory file name.
MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
MaximumVelocity	double *	Maximum velocity (units/s).
MaximumAcceleration	double *	Maximum acceleration (units/s <sup>2</sup> ).

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.1.355 XYMappingGet

#### Name

**XYMappingGet** – Read data of a line of an XY mapping matrix in the controller.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name (must be a XY positioner): (-18)
- Check the secondary positioner (must not be a secondary positioner): (-24)
- Check MappingNumber (must be  $> 0$  and  $\leq 1$  (standard firmware) or 3 (Precision Platform firmware)): (-9)
- Check LineNumber (must be  $\geq 1$  and  $\leq$  MappingLineNumber): (-9)
- Check the number of parameters (must be  $> 3$  and  $\leq$  MappingColumnNumber+3): (-9)
- Check mapping enable state (must be Enabled): (-121)

#### Description

Read data of a line of a XY mapping matrix in the controller.

#### *Example:*

0	-3.00	-2.00	-1.00	0.00	1.00	2.00	3.00	
-3.00	-0.00192	-0.00534	-0.00254	0.00023	0.00254	0.00534	0.00192	(Line 1)
-2.00	-0.00453	-0.00322	-0.00676	0.00049	0.00676	0.00322	0.00453	
-1.00	-0.00331	-0.00845	-0.00769	0.00102	0.00769	0.00845	0.00331	
0.00	-0.00787	-0.00228	-0.00787	0	0.00787	0.00228	0.00787	
1.00	-0.00232	-0.00210	-0.00342	0.00089	0.00342	0.00210	0.00232	
2.00	-0.00134	-0.00308	-0.00675	0.00101	0.00675	0.00308	0.00134	
3.00	-0.00789	-0.00148	-0.00234	0.00121	0.00234	0.00148	0.00789	

#### Prototype

```
int XYMappingGet(
    int SocketID,
    char * PositionerName,
    int* MappingNumber,
    int* LineNumber,
    double* Value1,
    double* Value2,
    ...
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	XY positioner name.
MappingNumber	int	XY mapping matrix number.
LineNumber	int	Mapping line to read.

### **Output parameters**

Value1	double *	Data of column #1 of line # <i>LinerNumber</i> of the mapping file # <i>MappingNumber</i> .
Value2	double *	Data of column #2 of line # <i>LinerNumber</i> of the mapping file # <i>MappingNumber</i> .
...	double *	One data per column of of line # <i>LinerNumber</i> of the mapping file # <i>MappingNumber</i> .

### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -9: Wrong number of parameters in the command.
- -17: Parameter out of range or incorrect.
- -18: PositionerName doesn't exist or unknown command.
- -24: Not available in this configuration (secondary positioner is not allowed).
- -121: Function is not allowed due to mapping disabled.

### 6.2.1.356 XYMappingSet

#### Name

**XYMappingSet** – Change data of a line of an XY mapping matrix in the controller.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name (must be a XY positioner): (-18)
- Check the secondary positioner (must not be a secondary positioner): (-24)
- Check MappingNumber (must be  $> 0$  and  $\leq 1$  (standard firmware) or 3 (Precision Platform firmware)): (-9)
- Check LineNumber (must be  $\geq 1$  and  $\leq$  MappingLineNumber): (-9)
- Check the number of parameters (must be  $> 3$  and  $\leq$  MappingColumnNumber+3): (-9)
- Check the group status (must be NOTINIT or DISABLE): (-22)
- Check mapping enable state (must be Enabled): (-121)
- Check group state (must be NOTINIT, DISABLE or READY at HomePreset position): (-205)

#### Description

Change data of a line of a XY mapping matrix.

It's possible to execute this function only when the XY group is in one of following states (*otherwise API returns error -205*) :

- NOTINIT
- DISABLE
- READY and all positioners (X and Y) are at HomePreset position.

#### *Example:*

0	-3.00	-2.00	-1.00	0.00	1.00	2.00	3.00	
-3.00	-0.00192	-0.00534	-0.00254	0.00023	0.00254	0.00534	0.00192	(Line 1)
-2.00	-0.00453	-0.00322	-0.00676	0.00049	0.00676	0.00322	0.00453	
-1.00	-0.00331	-0.00845	-0.00769	0.00102	0.00769	0.00845	0.00331	
0.00	-0.00787	-0.00228	-0.00787	0	0.00787	0.00228	0.00787	
1.00	-0.00232	-0.00210	-0.00342	0.00089	0.00342	0.00210	0.00232	
2.00	-0.00134	-0.00308	-0.00675	0.00101	0.00675	0.00308	0.00134	
3.00	-0.00789	-0.00148	-0.00234	0.00121	0.00234	0.00148	0.00789	

**Prototype**

```
int XYMappingSet(
    int SocketID,
    char * PositionerName,
    int MappingNumber,
    int LineNumber,
    double Value1,
    double Value2,
    ...
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	XY positioner name.
MappingNumber	int	XY mapping matrix number.
LineNumber	int	Mapping line to change.
Value1	double	Data of column #1 of line # <i>LineNumber</i> of the mapping file # <i>MappingNumber</i>
Value2	double	Data of column #2 of line # <i>LineNumber</i> of the mapping file # <i>MappingNumber</i>
...	double	One data per column of of line # <i>LineNumber</i> of the mapping file # <i>MappingNumber</i>

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -9: Wrong number of parameters in the command.
- -17: Parameter out of range or incorrect.
- -18: PositionerName doesn't exist or unknown command.
- -22: Not allowed action
- -24: Not available in this configuration (secondary positioner is not allowed).
- -121: Function is not allowed due to mapping disabled.
- -205: Not enable in your configuration.

### 6.2.1.357 XYPTExecution

#### Name

**XYPTExecution** – Execute a PT trajectory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length: (-3)
- Check group type (must be a XY group): (-8)
- Check input value (number of executions must >0): (-17)
- Check group name: (-19)
- Group state must be “READY”: (-22)
- Check backlash (must not be enabled): (-46)
- Check BaseVelocity (stages.ini, must = 0): (-48)
- Check trajectory file existence or file reading: (-61)
- Check message queue: (-71)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function executes a PT (Position Time) trajectory. The trajectory file must be stored in the folder “\Admin\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (message queue or task error) then (-72) is returned.

Before a trajectory execution, it is recommended to check whether the trajectory is within the positioner motion capabilities by using “XYPTVerification” and “XYPTVerificationResultGet” functions.

During the trajectory execution, if a positioner reaches one of travel limits, the trajectory execution will stop and the (-25) error is generated in the positioner errors.

---

#### NOTES

**In case of an (-33) error, an (-25) error or (-44) error, the group state becomes DISABLE. To help determine the error source, check the positioner errors, the hardware status and the driver status.**

**This function can be used only with the XPS-Q Precision Platform controller.**

---

#### Prototype

```
int XYPTExecution(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250],  
    int ExecutionNumber  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
ExecutionNumber	int	Number of trajectory executions.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.
- -25: Following Error.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -46: Not allowed action due to backlash.
- -48: BaseVelocity must be null.
- -61: Error file corrupt or file doesn't exist.
- -71: Error read from or write in message queue.
- -72: Error trajectory initialization.

### 6.2.1.358 **XYPTLoadToMemory**

#### Name

**XYPTLoadToMemory** – Load some lines of PT trajectory to the controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function loads some lines of PT trajectory into XPS controller memory. Each trajectory element must be separated by a comma. The trajectory lines are separated between them by a “\n” (LF) character. To verify or to execute the PT trajectory loaded in memory, use the string “**FromMemory**” instead of a file name.

#### NOTE

**All of the PT functions, when called with the string “FromMemory” instead of a FileName, will perform the same operation as the PT trajectory in RAM as it does from a disk.**

#### Example:

```
XYPTLoadToMemory(socketId,myGroup,"dT1,dX11,dX12\n...dTn,dXn1,dXn2\n")
XYPTVerification (socketId,myGroup,FromMemory)
XYPTExecution(socketId,myGroup,FromMemory,1).
```

#### Prototype

```
int XYPTLoadToMemory(
    int SocketID,
    char GroupName[250],
    char TrajectoryData[400]
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
TrajectoryData	char *	Trajectory data lines.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.359 XYPTParametersGet

#### Name

**XYPTParametersGet** – Get PT trajectory parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Check current executing trajectory type (must be PT): (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function returns the PT trajectory parameters (trajectory name and current executing element number) of the current PT trajectory.

#### Prototype

```
int XYPTParametersGet(  
    int SocketID,  
    char GroupName[250],  
    char * FileName,  
    int * CurrentElementNumber  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

FileName	char *	Currently executing trajectory file name.
CurrentElementNumber	int *	Currently executing element number.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### **6.2.1.360 XYPTPulseOutputGet**

#### **Name**

**XYPTPulseOutputGet** – Get the configuration of pulse generation of a PT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function returns the last configuration of pulse generation of a PT trajectory, that was previously set by *XYPTPulseOutputSet()*.

The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

#### ***Example:***

*XYPTPulseOutputSet(MyGroup, 3, 5, 0.01)*

*XYPTPulseOutputGet(MyGroup) => 0,3,5,0.01 (0 is error return, means OK)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

Start element= 3

End element = 5

Time interval = 0.01 seconds.

#### **Prototype**

```
int XYPTPulseOutputGet(
    int SocketID,
    char GroupName[250],
    int * StartElement,
    int * EndElement,
    double * TimeInterval
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

StartElement	int *	Start pulse element number.
EndElement	int *	End pulse element number.
TimeInterval	double *	Time interval between pulses (seconds).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.361 **XYPTPulseOutputSet**

#### Name

**XYPTPulseOutputSet** – Set the configuration of pulse generation of a PT trajectory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Check the pulse generation must not be in progress: (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function configures and activates the pulse generation of a PT trajectory. The pulse generation is defined by a start element, an end element, and a time interval in seconds. If a pulse generation is already activated on the selected PT trajectory then this function returns -22 (“Not allowed action”) error.

Please note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, it is required to define the pulse output settings again.

This capability allows output of pulses at constant time intervals on a PT trajectory. The pulses are generated between the first and the last trajectory element. The minimum possible time interval is CorrectorISRPeriod value (*system.ref*).

The trajectory pulses are generated on the following GPIO outputs:

GPIO signals	ISA XPS controller	PCI XPS controller (for example XPS-RL) with basic GPIO board	PCI XPS controller (for example XPS-RL) with extended GPIO board
Window	GPIO2, pin 11	GPIO1.DO6	GPIO5.DO14
Pulses	GPIO2, pin 12	GPIO1.DO7	GPIO5.DO15

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual, Appendix / General I/O Description.

#### Example:

*XYPTPulseOutputSet(Group1, 3, 5, 0.01)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

#### Prototype

```
int XYPTPulseOutputSet(
    int SocketID,
    char GroupName[250],
    int StartElement,
    int EndElement,
    double TimeInterval
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
StartElement	int	Start pulse element number.
EndElement	int	End pulse element number.
TimeInterval	double	Time interval between pulses (seconds).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### 6.2.1.362 XYPTResetInMemory

#### Name

**XYPTResetInMemory** – Deletes the content of the PT trajectory buffer in the controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function deletes the PT trajectory buffer in the controller memory, that was previously loaded with the “XYPTLoadToMemory” function.

#### Prototype

```
int XYPTResetInMemory(  
    int SocketID,  
    char GroupName[250]  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.363 XYPTVerification

#### Name

**XYPTVerification** – Check a PT trajectory data file.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length (must  $\leq 250$ ): (-3)
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Check BaseVelocity value (must = 0): (-48)
- Check trajectory file existence and the file format: (-61)
- Check trajectory (number of elements must  $> 0$ ): (-66)
- Check velocity (Minimum Velocity  $\leq$  Velocity  $\leq$  Maximum Velocity): (-68)
- Check acceleration (Minimum acc.  $\leq$  acceleration  $\leq$  Maximum acc.): (-69)
- Check end output velocity (must = 0): (-70)
- Check delta time (DeltaTime must  $> 0$ ): (-75)

#### Description

This function verifies the execution of a PT trajectory. The results of the verification can be got with the “XYPTVerificationResultGet” function. The trajectory file must be stored in the folder “\ADMIN\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (task error) then the (-72) error is returned.

- This function can be executed at any time and is independent of the trajectory execution. It performs the following:
- Checks the trajectory file for data coherence.
- Calculates the trajectory limits, which are: the required travel per positioner, the maximum possible trajectory velocity and the maximum possible trajectory acceleration. This function helps define the parameters for the trajectory execution.
- The required travel values (MinimumPosition and MaximumPosition) are calculated relative to the position zero, not to the current position. So before executing a PT trajectory, the user must pay attention to the current position of the positioners to make sure that the trajectory will not exceed the positioner travel limits.
- If all is OK, it returns “SUCCESS” (0). Otherwise, it returns a corresponding error.

---

#### NOTES

**Because of the PT trajectory internal calculation of elements end velocity, a correct PT trajectory file must have at least two lines with zero displacements at the trajectory end. Otherwise, the “XYPTVerification” function returns the (-70) error.**

**The “XYPTVerification” function is independent from the “XYPTExecution” function, but it is highly recommended to execute this function before executing a PT trajectory.**

**This function can be used only with the XPS-Q Precision Platform controller.**

---

### **Prototype**

```
int XYPTVerification(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250]  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.

### **Output parameters**

None.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.
- -61: Error file corrupt or file doesn't exist.
- -66: Trajectory doesn't content any element.
- -68: Acceleration on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -70: Final velocity on trajectory is not zero.
- -72: Error trajectory initialization.
- -75: Trajectory element has a negative or null delta T.

### 6.2.1.364 XYPTVerificationResultGet

#### Name

**XYPTVerificationResultGet** – Get the results of the “XYPTVerification” function.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name length (must  $\leq 250$ ): (-3)
- Check positioner name: (-18)
- Check the last XY PTVerification (must be done): (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function returns the results of the previous “XYPTVerification” function, positioner by positioner. The results are the travel requirements (min and max values), the possible maximum velocity and the possible maximum acceleration.

If no verification was previously done then the (-22) error is returned.

#### Prototype

```
int XYPTVerificationResultGet(  
    int SocketID,  
    char PositionerName[250],  
    char * TrajectoryFileName,  
    double * MinimumPosition,  
    double * MaximumPosition,  
    double * MaximumVelocity,  
    double * MaximumAcceleration  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

TrajectoryFileName	char *	Examined trajectory file name.
MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
MaximumVelocity	double *	Maximum velocity (units/s).
MaximumAcceleration	double *	Maximum acceleration (units/s <sup>2</sup> ).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### 6.2.1.365 XYPVTExecution

#### Name

**XYPVTExecution** – Execute a PVT trajectory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length: (-3)
- Check group type (must be a XY group): (-8)
- Check input value (number of executions must >0): (-17)
- Check group name: (-19)
- Group state must be "READY": (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Check backlash (must not be enabled): (-46)
- Check BaseVelocity (stages.ini, must = 0): (-48)
- Check trajectory file existence or file reading: (-61)
- Check message queue: (-71)

#### Description

This function executes a PVT (Position Velocity Time) trajectory. The trajectory file must be stored in the folder “\Admin\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (message queue or task error) then (-72) is returned.

Before a trajectory execution, it is recommended to check whether the trajectory is within the positioner motion capabilities by using “XYPVTVerification” and “XYPVTVerificationResultGet” functions.

During the trajectory execution, if a positioner reaches one of travel limits, the trajectory execution will stop and the (-25) error is generated in the positioner errors.

---

#### NOTES

**In case of an (-33) error, an (-25) error or (-44) error, the group state becomes DISABLE. To help determine the error source, check the positioner errors, the hardware status and the driver status.**

---

#### Prototype

```
int XYPVTExecution(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250],  
    int ExecutionNumber  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
ExecutionNumber	int	Number of trajectory executions.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.
- -25: Following Error.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -46: Not allowed action due to backlash.
- -48: BaseVelocity must be null.
- -61: Error file corrupt or file doesn't exist.
- -71: Error read from or write in message queue.
- -72: Error trajectory initialization.

### 6.2.1.366 XYPVTLoadToMemory

#### Name

**XYPVTLoadToMemory** – Load some lines of PVT trajectory to the controller memory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function loads some lines of PVT trajectory into XPS controller memory. Each trajectory element must be separated by a comma. The trajectory lines are separated between them by a “\n” (LF) character. To verify or to execute the PVT trajectory loaded in memory, use the string “**FromMemory**” instead of a file name.

#### NOTE

All of the PVT functions, when called with the string “FromMemory” instead of a FileName, will perform the same operation as the PVT trajectory in RAM as it does from a disk.

#### Example:

```
XYPVTLoadToMemory(socketId,myGroup,"dT1,dX11,Vout11,dX12,Vout12|n...dTn,dXn1,Voutn1,dXn2,Voutn2|n")
XYPVTVerification (socketId,myGroup,FromMemory)
XYPVTExecution(socketId,myGroup,FromMemory,I).
```

#### Prototype

```
int XYPVTLoadToMemory(
    int SocketID,
    char GroupName[250],
    char TrajectoryData[400]
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
TrajectoryData	char *	Trajectory data lines.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### 6.2.1.367 XYPVTParametersGet

#### Name

**XYPVTParametersGet** – Get PVT trajectory parameters.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Check current executing trajectory type (must be PVT): (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function returns the PVT trajectory parameters (trajectory name and current executing element number) of the current PVT trajectory.

#### Prototype

```
int XYPVTParametersGet(  
    int SocketID,  
    char GroupName[250],  
    char * FileName,  
    int * CurrentElementNumber  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

FileName	char *	Currently executing trajectory file name.
CurrentElementNumber	int *	Currently executing element number.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### 6.2.1.368 XYPVTPulseOutputGet

#### Name

**XYPVTPulseOutputGet** – Get the configuration of pulse generation of a PVT trajectory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function returns the last configuration of pulse generation of a PVT trajectory, that was previously set by *XYPVTPulseOutputSet()*.

The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

#### *Example:*

*XYPVTPulseOutputSet(MyGroup, 3, 5, 0.01)*

*XYPVTPulseOutputGet(MyGroup) => 0,3,5,0.01 (0 is error return, means OK)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

Start element= 3

End element = 5

Time interval = 0.01 seconds.

#### Prototype

```
int XYPVTPulseOutputGet(
    int SocketID,
    char GroupName[250],
    int * StartElement,
    int * EndElement,
    double * TimeInterval
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### Output parameters

StartElement	int *	Start pulse element number.
EndElement	int *	End pulse element number.
TimeInterval	double *	Time interval between pulses (seconds).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### **6.2.1.369 XYPVTPulseOutputSet**

#### **Name**

**XYPVTPulseOutputSet** – Set the configuration of pulse generation of a PVT trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Check the pulse generation must not be in progress: (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function configures and activates the pulse generation of a PVT trajectory. The pulse generation is defined by a start element, an end element, and a time interval in seconds. If a pulse generation is already activated on the selected PVT trajectory then this function returns -22 (“Not allowed action”) error.

Please note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, it is required to define the pulse output settings again.

This capability allows output of pulses at constant time intervals on a PVT trajectory. The pulses are generated between the first and the last trajectory element. The minimum possible time interval is CorrectorISRPeriod value (*system.ref*).

The trajectory pulses are generated on the following GPIO outputs:

GPIO signals	ISA XPS controller	PCI XPS controller (for example XPS-RL) with basic GPIO board	PCI XPS controller (for example XPS-RL) with extended GPIO board
Window	GPIO2, pin 11	GPIO1.DO6	GPIO5.DO14
Pulses	GPIO2, pin 12	GPIO1.DO7	GPIO5.DO15

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual, Appendix / General I/O Description.

#### ***Example:***

*XYPVTPulseOutputSet(Group1, 3, 5, 0.01)*

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

#### **Prototype**

```
int XYPVTPulseOutputSet(
    int SocketID,
    char GroupName[250],
    int StartElement,
    int EndElement,
    double TimeInterval
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
StartElement	int	Start pulse element number.
EndElement	int	End pulse element number.
TimeInterval	double	Time interval between pulses (seconds).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### **6.2.1.370 XYPVTResetInMemory**

#### **Name**

**XYPVTResetInMemory** – Deletes the content of the PVT trajectory buffer in the controller memory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### **Description**

This function deletes the PVT trajectory buffer in the controller memory, that was previously loaded with the “XYPVTLoadToMemory” function.

#### **Prototype**

```
int XYPVTLoadToMemory(
    int SocketID,
    char GroupName[250],
    char FileName[250],
    char TrajectoryData[400]
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
TrajectoryData	char *	Trajectory data lines.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.

### **6.2.1.371 XYPVTVerification**

#### **Name**

**XYPVTVerification** – Check a PVT trajectory data file.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length (must  $\leq 250$ ): (-3)
- Check group type (must be a XY group): (-8)
- Check group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Check BaseVelocity value (must = 0): (-48)
- Check trajectory file existence and the file format: (-61)
- Check trajectory (number of elements must  $> 0$ ): (-66)
- Check velocity (Minimum Velocity  $\leq$  Velocity  $\leq$  Maximum Velocity): (-68)
- Check acceleration (Minimum acc.  $\leq$  acceleration  $\leq$  Maximum acc.): (-69)
- Check end output velocity (must = 0): (-70)
- Check delta time (DeltaTime must  $> 0$ ): (-75)

#### **Description**

This function verifies the execution of a PVT trajectory. The results of the verification can be got with the “XYPVTVerificationResultGet” function. The trajectory file must be stored in the folder “\ADMIN\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (task error) then the (-72) error is returned.

This function can be executed at any time and is independent of the trajectory execution. It performs the following:

- Checks the trajectory file for data coherence.
- Calculates the trajectory limits, which are: the required travel per positioner, the maximum possible trajectory velocity and the maximum possible trajectory acceleration. This function helps define the parameters for the trajectory execution.
- The required travel values (MinimumPosition and MaximumPosition) are calculated relative to the position zero, not to the current position. So before executing a PVT trajectory, the user must pay attention to the current position of the positioners to make sure that the trajectory will not exceed the positioner travel limits.
- If all is OK, it returns “SUCCESS” (0). Otherwise, it returns a corresponding error.

---

#### **NOTE**

The “XYPVTVerification” function is independent from the “XYPVTExecution” function, but it is highly recommended to execute this function before executing a PVT trajectory.

---

#### **Prototype**

```
int XYPVTVerification(
    int SocketID,
    char GroupName[250],
    char FileName[250]
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -24: Not available in this configuration.
- -61: Error file corrupt or file doesn't exist.
- -66: Trajectory doesn't content any element.
- -68: Velocity on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -70: Final velocity on trajectory is not zero.
- -72: Error trajectory initialization.
- -75: Trajectory element has a negative or null delta T.

### 6.2.1.372 XYPVTVerificationResultGet

#### Name

**XYPVTVerificationResultGet** – Get the results of the “XYPVTVerification” function.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name length (must  $\leq 250$ ): (-3)
- Check positioner name: (-18)
- Check the last XY PVTVerification (must be done): (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)

#### Description

This function returns the results of the previous “XYPVTVerification” function, positioner by positioner. The results are the travel requirements (min and max values), the possible maximum velocity and the possible maximum acceleration.

If no verification was previously done then the (-22) error is returned.

#### Prototype

```
int XYPVTVerificationResultGet(  
    int SocketID,  
    char PositionerName[250],  
    char * TrajectoryFileName,  
    double * MinimumPosition,  
    double * MaximumPosition,  
    double * MaximumVelocity,  
    double * MaximumAcceleration  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### Output parameters

TrajectoryFileName	char *	Examined trajectory file name.
MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
MaximumVelocity	double *	Maximum velocity (units/s).
MaximumAcceleration	double *	Maximum acceleration (units/s <sup>2</sup> ).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.
- -24: Not available in this configuration.

### 6.2.1.373 XYZGroupPositionCorrectedProfilerGet

#### Name

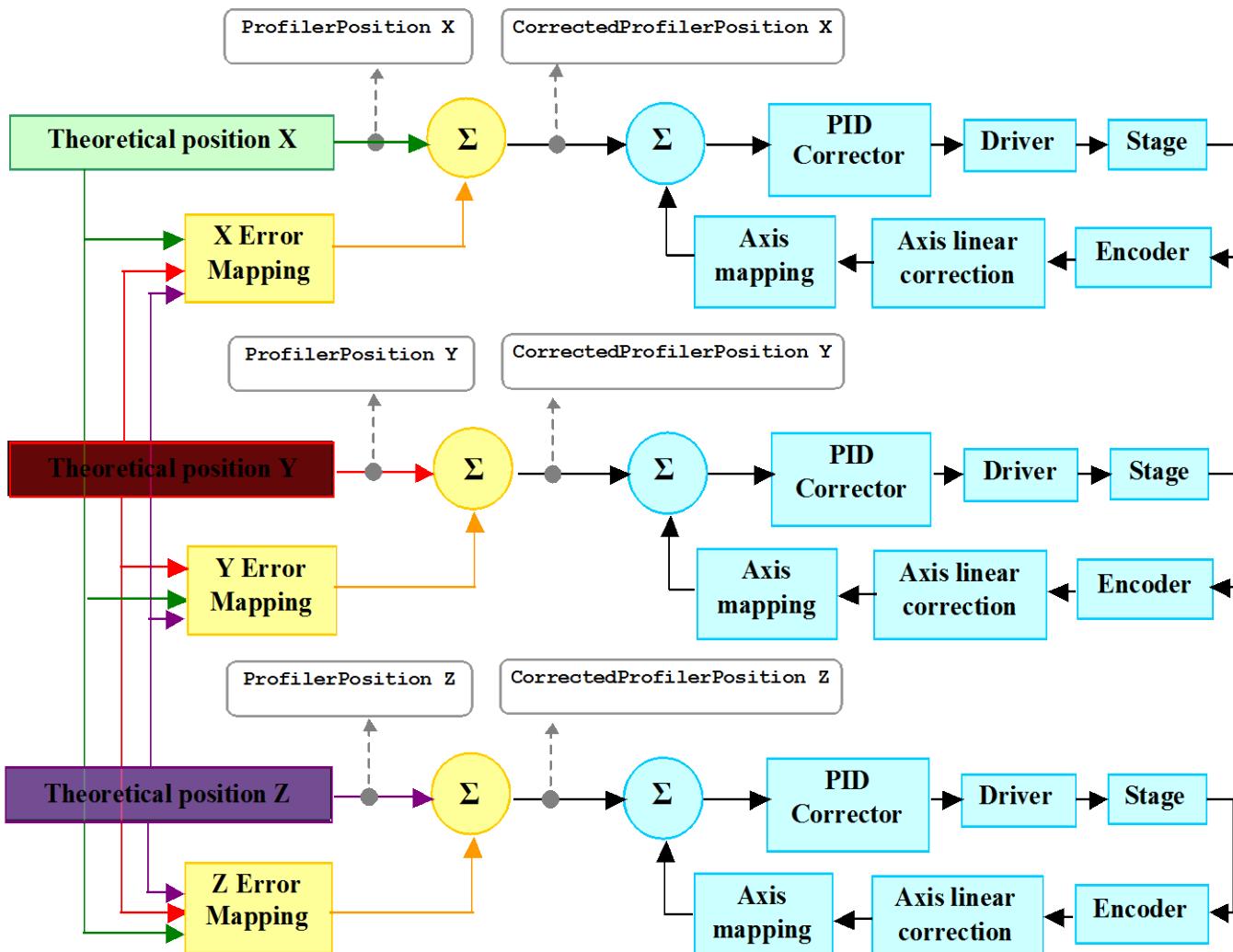
**XYZGroupPositionCorrectedProfilerGet** – Returns the corrected profiler position for all positioners of an XYZ group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Valid group type (must be an XYZ group): (-18)
- Valid group name: (-19)

#### Description

This function corrects a theoretical position which is recalculated with the XYZ mapping correction.



This function applies the XYZ mapping on the theoretical user positions and returns the corrected positions. These corrected profiler positions (X, Y and Z) take the XYZ mapping correction into account.

#### NOTE

This function is only allowed with an XYZ group.

### **Prototype**

```
int XYZGroupPositionCorrectedProfilerGet(  
    int SocketID,  
    char * GroupName,  
    char * FileName,  
    double PositionX,  
    double PositionY,  
    double PositionZ,  
    double * CorrectedPositionX,  
    double * CorrectedPositionY,  
    double * CorrectedPositionZ  
)
```

### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	XYZ group name.
PositionX	double	Theoretical position X.
PositionY	double	Theoretical position Y.
PositionZ	double	Theoretical position Z.

### **Output parameters**

CorrectedPositionX	double *	Corrected theoretical position.
CorrectedPositionY	double *	Corrected theoretical position Y.
CorrectedPositionZ	double *	Corrected theoretical position Z.

### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### **6.2.1.374 XYZGroupPositionPCORawEncoderGet**

#### **Name**

**XYZGroupPositionPCORawEncoderGet** – Returns the PCO raw encoder positions of an XYZ group.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type: (-8)
- Valid group type (must be a XYZ group): (-18)
- Valid group name: (-19)

#### **Description**

This function returns the X, Y and Z PCO raw encoder positions from the X, Y and Z user positions.

#### **NOTE**

**This function is only allowed with a XYZ group.**

#### **Prototype**

```
int XYZGroupPositionPCORawEncoderGet(
    int SocketID,
    char * GroupName,
    double PositionX,
    double PositionY,
    double PositionZ,
    double * PCORawPositionX,
    double * PCORawPositionY,
    double * PCORawPositionZ
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	XYZ group name.
PositionX	double	User position X.
PositionY	double	User position Y.
PositionZ	double	User position Z.

#### **Output parameters**

PCORawPositionX	double *	PCO Raw position X.
PCORawPositionY	double *	PCO Raw position Y.
PCORawPositionZ	double *	PCO Raw position Z.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.

### 6.2.1.375 XYZSplineExecution

#### Name

**XYZSplineExecution** – Execute a Spline trajectory.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length: (-3)
- Check group type (must be a XYZ group): (-8)
- Check input value (Velocity and Acceleration must >0): (-17)
- Check group name: (-19)
- Group state must be "READY": (-22)
- Check backlash (must not be enabled): (-46)
- Check BaseVelocity (stages.ini, must = 0): (-48)
- Check trajectory file existence or file reading: (-61)
- Check message queue: (-71)
- Check the velocity (Velocity ≤ TrajectoryMaximumVelocity): (-68)
- Check the acceleration (Acceleration ≤ TrajectoryMaximumAcceleration): (-69)

#### Description

This function executes a Spline trajectory. The trajectory file must be stored in the folder “\Admin\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (message queue or task error) then (-72) is returned.

Before a trajectory execution, it is recommended to check whether the trajectory is within the positioner motion capabilities by using “XYZSplineVerification” and “XYZSplineVerificationResultGet” functions.

During the trajectory execution, if a positioner reaches one of travel limits, the trajectory execution will stop and the (-25) error is generated in the positioner errors.

---

#### NOTE

**In case of an (-33) error, an (-25) error or (-44) error, the group state becomes DISABLE. To help determine the error source, check the positioner errors, the hardware status and the driver status.**

---

#### Prototype

```
int XYZSplineExecution(  
    int SocketID,  
    char GroupName[250],  
    char FileName[250],  
    double Velocity,  
    double Acceleration  
)
```

### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.
Velocity	double	Trajectory velocity (units/s).
Acceleration	double	Trajectory acceleration (units/s <sup>2</sup> ).

### Output parameters

None.

- ***Return*** (*In addition to the results of testing the inputs of functions*)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -46: Not allowed action due to backlash.
- -48: BaseVelocity must be null.
- -61: Error file corrupt or file doesn't exist.
- -68: Velocity on trajectory is too big.
- -69: Acceleration on trajectory is too big.
- -71: Error read from or write in message queue.
- -72: Error trajectory initialization.

### **6.2.1.376 XYZSplineParametersGet**

#### **Name**

**XYZSplineParametersGet** – Get Spline trajectory parameters.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XYZ group): (-8)
- Check group name: (-19)
- Check current executing trajectory type (must be Spline): (-22)

#### **Description**

This function returns the Spline trajectory parameters (trajectory name, trajectory velocity, trajectory acceleration, current executing element number) of the current Spline trajectory.

#### **Prototype**

```
int XYZSplineParametersGet(
    int SocketID,
    char GroupName[250],
    char * FileName,
    double * Velocity,
    double * Acceleration,
    int * CurrentElementNumber
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.

#### **Output parameters**

FileName	char *	Currently executing trajectory file name.
Velocity	double *	Trajectory velocity (units/s).
Acceleration	double *	Trajectory acceleration (units/s <sup>2</sup> ).
CurrentElementNumber	int *	Currently executing element number.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.377 XYZSplinePulseOutputGet**

#### **Name**

**XYZSplinePulseOutputGet** – Get the configuration of pulse generation of a Spline trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XYZ group): (-8)
- Check group name: (-19)

#### **Description**

This function returns the last configuration of pulse generation of a Spline trajectory, that was previously set by *XYZSplinePulseOutputSet()*.

The pulse output configuration is defined by a pulse start trajectory curved length, a pulse end trajectory curved length, and a pulse trajectory curved length interval.

#### ***Example:***

*XYZSplinePulseOutputSet(MyGroup, 10, 30, 0.01)*

*XYZSplinePulseOutputGet(MyGroup) => 0,10,30,0.01 (0 is error return, means OK)*

One pulse will be generated every 10 µm on the Spline trajectory between 10 mm and 30 mm trajectory curved lengths.

Pulse start trajectory curved length = 10 mm

Pulse end trajectory curved length = 30 mm

Pulse trajectory curved length interval = 0.01 mm.

#### **Prototype**

```
int XYZSplinePulseOutputGet(
    int SocketID,
    char GroupName[250],
    double * StartLength,
    double * EndLength,
    double * PathLengthInterval
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

GroupName	char *	Group name.
-----------	--------	-------------

#### ***Output parameters.***

StartLength	double *	Pulse start length (units).
-------------	----------	-----------------------------

EndLength	double *	Pulse end length (units).
-----------	----------	---------------------------

PathLengthInterval	double *	Pulse length interval (units).
--------------------	----------	--------------------------------

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.

### **6.2.1.378 XYZSplinePulseOutputSet**

#### **Name**

**XYZSplinePulseOutputSet** – Set the configuration of pulse generation of a Spline trajectory.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check group type (must be a XYZ group): (-8)
- Check group name: (-19)
- Check the pulse generation must not be in progress: (-22)

#### **Description**

This function configures and activates the pulse generation of a Spline trajectory. The pulse generation is defined by a start element, an end element, and a time interval in seconds. If a pulse generation is already activated on the selected Spline trajectory then this function returns -22 (“Not allowed action”) error.

Please note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, it is required to define the pulse output settings again.

This capability allows to generate pulses at constant pulse trajectory curved length intervals on a Spline trajectory. The pulses are generated between a pulse start trajectory curved length and a pulse end trajectory curved length. All lengths are calculated in an orthogonal XYZ coordination system.

The trajectory pulses are generated on the following GPIO outputs:

<b>GPIO signals</b>	<b>ISA XPS controller</b>	<b>PCI XPS controller (for example XPS-RL) with basic GPIO board</b>	<b>PCI XPS controller (for example XPS-RL) with extended GPIO board</b>
Window	GPIO2, pin 11	GPIO1.DO6	GPIO5.DO14
Pulses	GPIO2, pin 12	GPIO1.DO7	GPIO5.DO15

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual, Appendix / General I/O Description.

#### ***Example:***

*XYZSplinePulseOutputSet(Group1, 10, 30, 0.01)*

One pulse will be generated every 10 µm on the Spline trajectory between 10 mm and 30 mm trajectory curved lengths.

Pulse start trajectory curved length = 10 mm

Pulse end trajectory curved length = 30 mm

Pulse trajectory curved length interval = 0.01 mm

**Prototype**

```
int XYZSplinePulseOutputSet(
    int SocketID,
    char GroupName[250],
    double StartLength,
    double EndLength,
    double PathLengthInterval
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
StartLength	double	Pulse start length (units).
EndLength	double	Pulse end length (units).
PathLengthInterval	double	Pulse length interval (units).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.

### **6.2.1.379 XYZSplineVerification**

#### **Name**

**XYZSplineVerification** – Check a Spline trajectory data file.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check trajectory file name length (must  $\leq 250$ ): (-3)
- Check group type (must be a XYZ group): (-8)
- Check group name: (-19)
- Check trajectory file existence and the file format: (-61)
- Check trajectory (number of elements must  $> 0$ ): (-66)

#### **Description**

This function verifies the execution of a Spline trajectory. The results of the verification can be got with the “XYZSplineVerificationResultGet” function. The trajectory file must be stored in the folder “\ADMIN\Public\Trajectory” of the XPS controller. If the trajectory cannot be initialized (task error) then the (-72) error is returned.

This function can be executed at any time and is independent of the trajectory execution. It performs the following:

- Checks the trajectory file for data coherence.
- Calculates the trajectory limits, which are: the required travel per positioner, the maximum possible trajectory velocity and the maximum possible trajectory acceleration. This function helps define the parameters for the trajectory execution.
- The required travel values (MinimumPosition and MaximumPosition) are calculated relative to the position zero, not to the current position. So before executing a Spline trajectory, the user must pay attention to the current position of the positioners to make sure that the trajectory will not exceed the positioner travel limits.
- If all is OK, it returns “SUCCESS” (0). Otherwise, it returns a corresponding error.

#### **NOTE**

The “XYZSplineVerification” function is independent from the “XYZSplineExecution” function, but it is highly recommended to execute this function before executing a Spline trajectory.

#### **Prototype**

```
int XYZSplineVerification(
    int SocketID,
    char GroupName[250],
    char FileName[250]
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	Group name.
FileName	char *	Trajectory file name.

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -3: String too long.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -61: Error file corrupt or file doesn't exist.
- -66: Trajectory doesn't contain any element.
- -72: Error trajectory initialization.

### **6.2.1.380 XYZSplineVerificationResultGet**

#### **Name**

**XYZSplineVerificationResultGet** – Get the results of the “XYZSplineVerification” function.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check positioner name length (must  $\leq 250$ ): (-3)
- Check positioner name: (-18)
- Check the last XYZ SplineVerification (must be done): (-22)

#### **Description**

This function returns the results of the previous “XYZSplineVerification” function, positioner by positioner. The results are the travel requirements (min and max values), the possible maximum velocity and the possible maximum acceleration.

If no verification was previously done then the (-22) error is returned.

#### **Prototype**

```
int XYZSplineVerificationResultGet(
    int SocketID,
    char PositionerName[250],
    char * TrajectoryFileName,
    double * MinimumPosition,
    double * MaximumPosition,
    double * MaximumVelocity,
    double * MaximumAcceleration
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.

#### **Output parameters**

TrajectoryFileName	char *	Examined trajectory file name.
MinimumPosition	double *	Minimum position (units).
MaximumPosition	double *	Maximum position (units).
MaximumVelocity	double *	Maximum velocity (units/s).
MaximumAcceleration	double *	Maximum acceleration (units/s <sup>2</sup> ).

#### **Return (In addition to the results of testing the inputs of functions)**

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn't exist or unknown command.
- -22: Not allowed action.

## 6.2.2 Extended Functions

The functions described in this section are for a specific controller configuration, please contact Newport for further information.

### 6.2.2.1 AbortMove

#### Name

**AbortMove** – abort the motion or the jog in progress for the XY group.

#### Input tests

- Refer to section 6.1: "Input Tests Common to all XPS Functions".

#### Description

This function allows aborting a motion or a jog in progress. The group status must be "MOVING" or "JOGGING" else the (-22) error is returned.

If the group status is "MOVING", this function stops all motion in progress.

If the group status is "JOGGING", this function stops all "jog" motion in progress and disables the jog mode. After this "group move abort" action, the group status becomes "READY".

#### Prototype

```
int AbortMove(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the "TCP_ConnectToServer" function.
----------	-----	---

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -27: Move Aborted.

### 6.2.2.2 EndJog

#### Name

**EndJog** – Disables the jog mode in the XY group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Check the group name: (-19)
- Group status must be “READY”: (-22)

#### Description

This function disables the Jog mode for the first declared XY group. To use this function, the group must be in the “JOGGING” state and all positioners must be idle (means velocity must be 0).

This function allows to exit the “JOGGING” state and to come back to the “READY” state. If the group state is not “JOGGING” or if the profiler velocity is not null then the (-22) error is returned.

---

#### NOTE

**Use the “StartJog” function to enable the jog mode.**

---

#### Prototype

```
int GetJogAcceleration(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Not allowed due to a positioner error or hardware status.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.3 GetAccParams

#### Name

**GetAccParams** – Get acceleration parameters for X and Y axes.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This API gets the current acceleration parameters for X and Y axes.

The smooth factor represents the jerk time and all parameters unit in milliseconds.

#### Prototype

```
int GetAccParams(
```

```
    int SocketID,  
    int * XaccTime_ms,  
    int * XsmoothFactor_ms,  
    int * YaccTime_ms,  
    int * YsmoothFactor_ms  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

XaccTime_ms	int *	X acceleration time in msec.
XsmoothFactor_ms	int *	X Smooth factor in msec (jerk time).
YaccTime_ms	int *	Y acceleration time in msec.
YsmoothFactor_ms	int *	Y Smooth factor in msec (jerk time).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.2.4 GetBrakeState

#### Name

**GetBrakeState**– Get the brake status.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function reads the current brake command signal state from Inhibit or GPIO connector. Refer to section [BRAKE] in System.ini configuration file.

Brake command signal:

- 1) BrakeCommandSignalState parameter is defined “Direct” in System.ini
  - 0 = Brake OFF
  - 1 = Brake ON
- 2) BrakeCommandSignalState parameter is defined “Inverted” in System.ini
  - 0 = Brake ON
  - 1 = Brake OFF

#### Prototype

```
int GetBrakeState(  
    int SocketID,  
    int * BrakeStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

BrakeStatus	int *	The brake status.
-------------	-------	-------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.2.5 GetCurrentPosition

#### Name

**GetCurrentPosition** – Get all current positions of X, X2 and Y.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name is valid: (-18)
- Not allowed due to configuration disabled: (-121)
- Not allowed because group is not initialized or not referenced: (-135)

#### Description

Read all current positions of Y, X, X2, Y laser and X laser.

Units:

Y and X positions:  $\mu\text{m}$   
X1 and X2 positions: counts  
Y and X laser positions: counts

#### Prototype

```
int GetCurrentPosition (int SocketID, double * y_position_um, double *
x_position_um, int * x1_position_cnts, int * x2_position_cnts, int *
y_laser_position_cnts, int * x_laser_position_cnts)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

y_position_um	double *	Y position in $\mu\text{m}$ .
x_position_um	double *	X position in $\mu\text{m}$ .
x1_position_cnts	int *	X1 position in counts.
x2_position_cnts	int *	X2 position in counts.
y_laser_position_cnts	int *	Y laser position in counts.
x_laser_position_cnts	int *	X laser position in counts.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -18: PositionerName doesn't exist or unknown command.
- -121: Not allowed due to configuration disabled.
- -135: Not allowed because group is not initialized or not referenced.

### 6.2.2.6 GetGantryMode

#### Name

**GetGantryMode** – Get Gantry mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the set option: (-17)
- Check the group name is valid: (-19)
- Verify gantry mode getting is allowed: (-22)

#### Description

Get the current gantry option. Three “gantry” options are available:

- Option0 =>Gantry standard.
- Option1 =>Gantry force balance.
- Option2 =>Gantry force balance with interferometer.

#### Prototype

```
int GetGantryMode(  
    int SocketID,  
    char * Option  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Option	char *	Option selection(Option0, Option1 or Option2).
--------	--------	--

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.7 GetJogAcceleration

#### Name

**GetJogAcceleration** – Returns the acceleration set by “SetJogAcceleration”.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)

#### Description

This function returns the acceleration setting by the user to use the jog mode for one positioner or for all positioners of the XY group.

#### Prototype

```
int GetJogAcceleration(  
    int SocketID,  
    int * XaccelerationTime_ms,  
    int * XsmoothFactor_ms,  
    int * YaccelerationTime_ms,  
    int * YsmoothFactor_ms  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
XaccelerationTime_ms	int *	User jog Acceleration time for X in ms.
XsmoothFactor_ms	int *	User jog Smooth factor for X in ms.
YaccelerationTime_ms	int *	User jog Acceleration time for Y in ms.
YsmoothFactor_ms	int *	User jog Smooth factor for Y in ms.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Not allowed due to a positioner error or hardware status.
- -19: GroupName doesn't exist or unknown command.

### 6.2.2.8 GetJogVelocity

#### Name

**GetJogVelocity** – Changes “on the fly” the velocity in the jog mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)

#### Description

This function returns jog velocities and the jog velocity acknowledge timeout in milliseconds used by the jog mode in the XY group.

#### Prototype

```
int GetJogVelocity(  
    int SocketID,  
    double * Xvelocity,  
    double * Yvelocity,  
    int * joystickAckTimeout_ms  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Xvelocity	double *	user jog velocity for X in $\mu\text{m/s}$ .
Yvelocity	double *	user jog velocity for Y in $\mu\text{m/s}$ .
joystickAckTimeout_ms	int *	user jog velocity acknowledge timeout in ms.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Not allowed due to a positioner error or hardware status.
- -19: GroupName doesn't exist or unknown command.

### 6.2.2.9 GetPistonState

#### Name

**GetPistonState** – Get current status of Piston and Lift Pin.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Read the current status of Piston and Lift Pin:

- Piston command status.
- Piston limit UP (Engaged).
- Piston limit DOWN (Released).
- Lift Pin UP.

Refer to section [PISTON] in System.ini configuration file.

- PistonEngagedSignalState = Direct or Inverted
- PistonReleasedSignalState = Direct or Inverted
- PistonCommandSignalState = Direct or Inverted

#### **Piston signals:**

		<b>PistonCommandSignalState</b>	
		<b>Direct</b>	<b>Inverted</b>
<b>Piston Command Status</b>	<b>0</b>	Released	Engaged
	<b>1</b>	Engaged	Released

		<b>PistonEngagedSignalState</b>	
		<b>Direct</b>	<b>Inverted</b>
<b>Piston Engaged Status</b>	<b>0</b>	Not Engaged	Engaged
	<b>1</b>	Engaged	Not Engaged

		<b>PistonReleasedSignalState</b>	
		<b>Direct</b>	<b>Inverted</b>
<b>Piston Released Status</b>	<b>0</b>	Not Released	Released
	<b>1</b>	Released	Not Released

<b>Lift Pin UP Status</b>	<b>0</b>	Not activated (Lift Pin is down)
	<b>1</b>	Activated (Lift Pin is up)

#### Prototype

```
int GetPistonState(
    int SocketID,
    int * CommandState,
    int * isEngaged,
    int * isReleased,
    int * LiftPinUPIInterlock
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

**Output parameters**

CommandState	int *	Status of the Piston command (0 or 1).
isEngaged	int *	Status of the Piston Engaged (0 or 1).
isReleased	int *	Status of the Piston Released (0 or 1).
LiftPinUPIInterlock	int *	Status of the Lift Pin UP (0 or 1).

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -7: Wrong format in the command string.
- -9: Wrong number of parameters in the command.
- -15: Wrong parameter type in the command string: int, short, int \* or short \* expected.

### 6.2.2.10 GetVarX

#### Name

**GetVarX** – Set new value for a specified parameter name from stages.ini for Y positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter value: (-17)
- Check positioner name: (-18)
- Verify the positioner is from a XY group: (-22)

#### Description

Get parameter value from stages.ini file for the X positioner.

#### Prototype

```
int GetVarX(  
    int SocketID,  
    char * ParameterName,  
    double * ParameterValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ParameterName	char *	Stages.ini parameter name.

#### Output parameters

ParameterValue	double *	The value to be set for the “ParameterName”.
----------------	----------	--

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.11 GetVarXSecondary

#### Name

**GetVarXSecondary** – Get the configured value of the parameter name from stages.ini for X secondary positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter value: (-17)
- Check positioner name: (-18)
- Verify the positioner is from a XY group: (-22)

#### Description

Get the configured value of the parameter name from stages.ini for X secondary positioner.

#### Prototype

```
int GetVarXSecondary(  
    int SocketID,  
    char * ParameterName,  
    double * ParameterValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ParameterName	char *	Stages.ini parameter name.

#### Output parameters

ParameterValue	double *	The value to be set for the “ParameterName”.
----------------	----------	--

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.12 GetVarY

#### Name

**GetVarY** – Get the configured value of the parameter name from stages.ini for Y positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter value: (-17)
- Check positioner name: (-18)
- Verify the positioner is from a XY group: (-22)

#### Description

Get the configured value of the parameter name from stages.ini for Y positioner.

#### Prototype

```
int GetVarY(  
    int SocketID,  
    char * ParameterName,  
    double * ParameterValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ParameterName	char *	Stages.ini parameter name.

#### Output parameters

ParameterValue	double *	The value to be set for the “ParameterName”.
----------------	----------	--

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.13 GetVelParams

#### Name

**GetVelParams** – Get current velocity for X and Y axes.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check motion profile: (-8)

#### Description

This API updates the current acceleration parameters for X and Y axes.

The smooth factor represents the jerk time and all parameters unit in milliseconds.

#### Prototype

```
int GetVelParams(  
    int SocketID,  
    double * Xvelocity,  
    double * Yvelocity  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Xvelocity	double *	X velocity in $\mu\text{m}/\text{s}$ .
Yvelocity	double *	Y velocity in $\mu\text{m}/\text{s}$ .

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.2.14 GetVerCommand

#### Name

**GetVerCommand** – Return firmware version.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the controller name and the firmware version number.

Example of returned version string: “XPS-Q8 Firmware V2.1.0”

- Controller name is XPS-Q8.
- Firmware version is V2.1.0.

#### Prototype

```
int GetVerCommand(  
    int SocketID,  
    char * Version  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Version	char *	The firmware version.
---------	--------	-----------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.2.15 GetXVelParams

#### Name

**GetXVelParams** – This API returns the current velocity parameter in  $\mu\text{m/s}$  for X axis.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check motion profile: (-8)

#### Description

This API returns the current velocity parameter in  $\mu\text{m/s}$  for X axis.

#### Prototype

```
int GetXVelParams(  
    int SocketID,  
    double * Xvelocity  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Xvelocity	double *	X velocity in $\mu\text{m/s}$ .
-----------	----------	---------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.2.16 GetYVelParams

#### Name

**GetYVelParams** – This API returns the current velocity parameter in  $\mu\text{m/s}$  for Y axis.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check motion profile: (-8)

#### Description

This API returns the current velocity parameter in  $\mu\text{m/s}$  for Y axis.

#### Prototype

```
int GetYVelParams(  
    int SocketID,  
    double * Yvelocity  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Yvelocity	double *	Y velocity in $\mu\text{m/s}$ .
-----------	----------	---------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.2.17 GetZone

#### Name

**GetZone** – Get current parameters of the defined circle zone.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name is valid: (-18)
- Not allowed due to configuration disabled: (-121)
- Not allowed because group is not initialized or not referenced: (-135)

#### Description

Get current parameters of the defined circle zone. A digital output is defined in the stage.ini file. It's always available after the home search. All parameters are defined in  $\mu\text{m}$ . The Radius and Hysteresis are defined to signal when going out the circle.

#### Prototype

```
int GetZone (int SocketID, double * x_center_um, double * y_center_um, double *
radius_um, double * hysteresis_um)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

x_center_um	double *	X center in $\mu\text{m}$ .
y_center_um	double *	Y center in $\mu\text{m}$ .
radius_um	double *	Radius in $\mu\text{s}$ .
hysteresis_um	double *	Hysteresis in $\mu\text{m}$ .

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -18: PositionerName doesn't exist or unknown command.
- -121: Not allowed due to configuration disabled.
- -135: Not allowed because group is not initialized or not referenced.

### 6.2.2.18 InitializeAndHomeX

#### Name

**InitializeAndHomeX** – Do home search on X axis.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the set option: (-17)
- Verify this command is allowed: (-22)
- Check state of physical ends of run: (-113)
- Check the opened socket is valid: (-200)
- Check this command is enabled for the current controller configuration: (-205)

#### Description

Initializes the motor, calibrates the encoder and activates the servo loop of each positioner of the XY group.

Next, performs a home search on the X positioner and configures the gantry mode used after homing.

Once the home search is finished with success, the group must be in “READY” state only if the Y positioner is already referenced. If it’s not the case, a home search on the Y positioner must be done.

To be “READY”, all axes must be referenced.

---

#### NOTE

The home search routine for each positioner is defined in the “stages.ini” file by the “HomeSearchSequenceType” key.

The home search time out is defined in the “stages.ini” file by the “HomeSearchTimeOut” key.

---

#### Prototype

```
int InitializeAndHomeX(  
    int SocketID,  
    char * Option  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Option	char *	Option selection(Option0, Option1 or Option2).

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.
- -25: Following Error.
- -28: Home search timeout.
- -33: Motion done timeout.
- -35: Position is outside of travel limits.
- -49: Inconsistent mechanical zero during home search.
- -50: Motor initialization error. Check InitializationAccelerationLevel, ScalingAcceleration, MaximumJerkTime, EncoderResolution or EncoderScalePitch.
- -113: Both ends of run activated.
- -200: Invalid socket.
- -202: Not allowed action due to an external motion interlock.
- -205: Not enable in your configuration.
- -208: Not allowed action because piston is engaged.
- -1004: Zygo signal is not present.

### 6.2.2.19 InitializeAndHomeXY

#### Name

**InitializeAndHomeXY** – Do home search on X then Y axis.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the set option: (-17)
- Verify this command is allowed: (-22)
- Check state of physical ends of run: (-113)
- Check the opened socket is valid: (-200)
- Check this command is enabled for the current controller configuration: (-205)

#### Description

This function initializes the motor, calibrates the encoder and activates the servo loop of each positioner of the XY group. Then, it performs a home search on X positioner and after on Y positioner. It configures the gantry mode used after homing.

Once the home search is finished with success, the group must be in “READY” state.

---

#### NOTE

The selected gantry option during the initialization phase is Option0.

The home search routine for each positioner is defined in the “stages.ini” file by the “HomeSearchSequenceType” key.

The home search time out is defined in the “stages.ini” file by the “HomeSearchTimeOut” key.

The home search sequence is defined in the “stages.ini” file by the “InitializationAndHomeSearchSequence”. The value must be “XthenY”.

---

#### Prototype

```
int InitializeAndHomeXY(  
    int SocketID,  
    char * Option  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Option	char *	Option selection(Option0, Option1 or Option2).

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.
- -25: Following Error.
- -26: Kill command or Emergency signal: check each positioners and each slave positioners, check that motion does not exceed software limits when combined with mapping and other features.
- -27: Move Aborted.
- -28: Home search timeout.
- -33: Motion done timeout.
- -35: Position is outside of travel limits.
- -44: Slave error disabling master.
- -49: Inconsistent mechanical zero during home search.
- -50: Motor initialization error. Check InitializationAccelerationLevel, ScalingAcceleration, MaximumJerkTime, EncoderResolution or EncoderScalePitch.
- -113: Both ends of run activated.
- -120: Warning following error during move with position compare enabled.
- -200: Invalid socket.
- -205: Not enable in your configuration.

### 6.2.2.20 InitializeAndHomeY

#### Name

**InitializeAndHomeY** – Do home search on Y axis.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Verify this command is allowed: (-22)
- Check state of physical ends of run: (-113)
- Check the opened socket is valid: (-200)
- Check this command is enabled for the current controller configuration: (-205)

#### Description

This function initializes the motor, calibrates the encoder and activates the servo loop of each positioner of the XY group. Then, it performs a home search on the Y positioner.

Once the home search is finished with success, the group must be in “READY” state only if the X positioner is already referenced. If it’s not the case, a home search on the X positioner must be done.

To be in “READY” state, all axes must be referenced.

---

#### NOTE

The home search routine for each positioner is defined in the “stages.ini” file by the “HomeSearchSequenceType” key.

The home search time out is defined in the “stages.ini” file by the “HomeSearchTimeOut” key.

---

#### Prototype

```
int InitializeAndHomeY(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	---

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -22: Not allowed action.
- -25: Following Error.
- -28: Home search timeout.
- -33: Motion done timeout.
- 35: Position is outside of travel limits.
- -49: Inconsistent mechanical zero during home search.
- -50: Motor initialization error. Check InitializationAccelerationLevel, ScalingAcceleration, MaximumJerkTime, EncoderResolution or EncoderScalePitch.
- -113: Both ends of run activated.
- -200: Invalid socket.
- -202: Not allowed action due to an external motion interlock.
- -205: Not enable in your configuration.
- -208: Not allowed action because piston is engaged.
- -1004: Zygo signal is not present.

### **6.2.2.21 MoveAbsolute**

#### **Name**

**MoveAbsolute** – Moves the stage to the end position. The positions are defined in um.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Verify target position in relation with the travel limits: (-17)
  - TargetPosition  $\geq$  MinimumTargetPosition.
  - TargetPosition  $\leq$  MaximumTargetPosition.
- Valid positioner name: (-18)
- Valid group name: (-19)
- Group status must be “READY” or “MOVING”: (-22)

#### **Description**

This function executes an absolute motion to go to a target XY position. The group state must be “READY” or “MOVING” else the (-22) error is returned. If the group is “READY” then the group status becomes “MOVING”.

Each “positioner” move refers to the acceleration, velocity, minimumTjerkTime and maximumTjerkTime as defined in the “Stages.ini” file or as redefined by the “PositionerSGammaParametersSet” function.

If a slave error or a following error is detected during the moving then (-25) or ERR\_SLAVE (-44) error is returned. In this case, the motion in progress is stopped and the group status becomes “DISABLE”.

If a “MotionDoneMode” is defined as “VelocityAndPositionWindowMotionDone” then an (-33) error can be returned if the time out (defined by “MotionDoneTimeout” in the stages.ini file) is reached before the motion done.

If “AbortMove” or “GroupMoveAbort” is done, an (-27) error is returned. In this case, the motion in progress is stopped and the group status becomes “READY”.

If a “GroupKill” command, an emergency brake or an emergency stop is occurred, an (-26) error is returned. In this case, the motion in progress is stopped and the group status becomes “NOT INITIALIZED”.

---

#### **NOTE**

**The asynchronous moves for positioners of the same group are possible through the use of different sockets to send the function.**

---

#### **Prototype**

```
int MoveAbsolute(
    int SocketID,
    double PositonAbsoluteX_um,
    double PositionAbsoluteY_um
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositonAbsoluteX_um	double	Target position in $\mu\text{m}$ for X axis.
PositonAbsoluteY_um	double	Target position in $\mu\text{m}$ for Y axis.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -25: Following Error.
- -26: Kill command or Emergency signal: check each positioners and each slave positioners, check that motion does not exceed software limits when combined with mapping and other features.
- -27: Move Aborted.
- -33: Motion done timeout.
- -44: Slave error disabling master.
- -120: Warning following error during move with position compare enabled.

### 6.2.2.22 MoveSlice

#### Name

**MoveSlice** – Execute a slice move on an XY group.

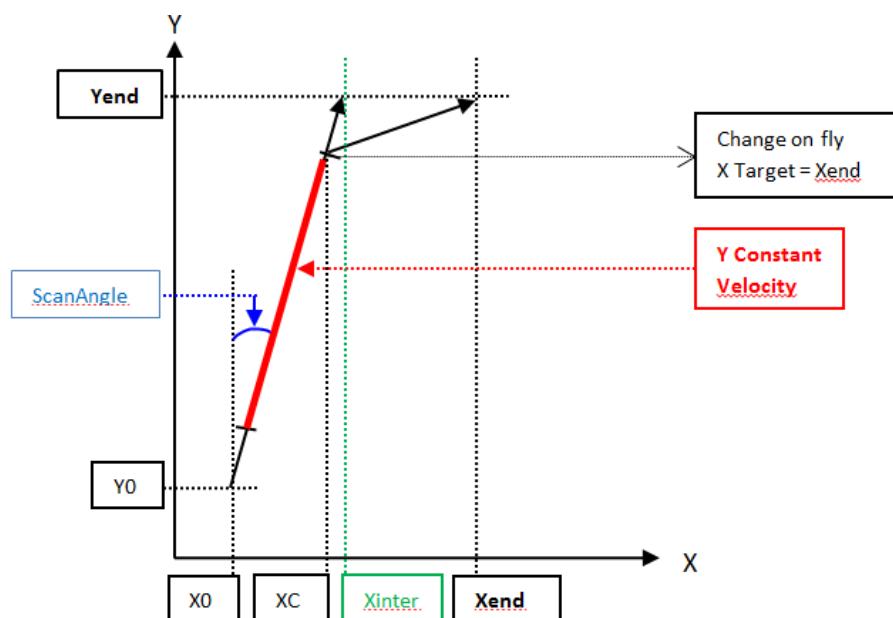
#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Verify target position in relation with the travel limits: (-17)
- Check the group status, it must be “READY” state: (-22)

#### Description

This API moves the stage to perform a “U-turn”.

The slice move begins a linear interpolation to move to (Xinter, Yend) but when the stage has finished the constant velocity phase, the slice move changes the target position to go to (Xend, Yend).



#### Prototype

```
int MoveSlice(
    int SocketID,
    double Yend_um,
    double Xend_um,
    double ScanAngle_urad
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Yend_um	double	y target position in $\mu\text{m}$ .
Xend_um	double	x target position in $\mu\text{m}$ .
ScanAngle_urad	double	Scan angle in $\mu\text{rad}$ .

### **Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -5: Not allowed due to a positioner error or hardware status.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.
- -202: Not allowed action due to an external motion interlock.
- -208: Not allowed action because piston is engaged.
- -1004: Zygo signal is not present.

### 6.2.2.23 RequestType1

#### Name

**RequestType1**– Get data collection Type 1.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the data collection of type1.

#### Prototype

```
int RequestType1(  
    int SocketID,  
    char * Header,  
    charhex32 * TimeStamp,  
    charhex32 * 1DFe,  
    charhex32 * 1DPos,  
    charhex32 * XFe,  
    charhex32 * XPos,  
    charhex32 * YawFe,  
    charhex32 * YawPos,  
    charhex32 * YFe,  
    charhex32 * YPos,  
    charhex32 * XMotor,  
    charhex32 * YMotor,  
    charhex32 * XSinCos,  
    charhex32 * YSinCos  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

### **Output parameters**

Header	char *      Type1, Type2 or Type3.
TimeStamp	charhex32 *   Internal counter in milliseconds.
1DFe	charhex32 *   1D+ following error in $\mu\text{m}$ .
1DPos	charhex32 *   1D+ current position in $\mu\text{m}$ .
XFe	charhex32 *   X following error in $\mu\text{m}$ .
XPos	charhex32 *   Yaw following error in $\mu\text{m}$ .
YawFe	charhex32 *   Yaw current position in $\mu\text{m}$ .
YFe	charhex32 *   Y following error in $\mu\text{m}$ .
YPos	charhex32 *   Y current position in $\mu\text{m}$ .
XMotor	charhex32 *   X motor current in 1/10000 full scale.
YMotor	charhex32 *   Y motor current in 1/10000 full scale.
XSinCos	charhex32 *   X Sin Cos in Volts.
YSinCos	charhex32 *   Y Sin Cos in Volts.

**Return** (In addition to the results of testing the inputs of functions)

- 0:      No error.
- -100:    Internal error (memory allocation error, ...).
- -136:    Wrong parameter type in the command string: charhex32 \* expected.

### 6.2.2.24 RequestType2

#### Name

**RequestType2**— Get data collection Type 2.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the data collection of type2.

#### Prototype

```
int RequestType2(  
    int SocketID,  
    char * Header,  
    charhex32 * TimeStamp,  
    charhex32 * ZygoLaserPower  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Header	char *	Type1, Type2 or Type3.
TimeStamp	charhex32 *	Internal counter in milliseconds.
ZygoLaserPower	charhex32 *	Zygo Laser Power (ON = 1, OFF = 0).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -136: Wrong parameter type in the command string: charhex32 \* expected.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.25 RequestType3

#### Name

**RequestType3**— Get data collection Type 3.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function returns the data collection of type3.

#### Prototype

```
int RequestType3(
    int SocketID,
    char * Header,
    charhex32 * TimeStamp,
    charhex32 * ZygоСignalStrength1,
    charhex32 * ZygоСignalStrength2
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Header	char *	Type1, Type2 or Type3.
TimeStamp	charhex32 *	Internal counter in milliseconds.
ZygоСignalStrength1	charhex32 *	Zygo Signal Strength 1.
ZygоСignalStrength2	charhex32 *	Zygo Signal Strength 2.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -17: Parameter out of range or incorrect.
- -136: Wrong parameter type in the command string: charhex32 \* expected.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.26 SetAccParams

#### Name

**SetAccParams** – Set acceleration parameters for X and Y axes.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This API updates the current acceleration parameters for X and Y axes.

The smooth factor represents the jerk time and all parameters unit in milliseconds.

#### Prototype

```
int SetAccParams(  
    int SocketID,  
    int XaccTime_ms,  
    int XsmoothFactor_ms,  
    int YaccTime_ms,  
    int YsmoothFactor_ms  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
XaccTime_ms	int	X acceleration time in msec.
XsmoothFactor_ms	int	X Smooth factor in msec (jerk time).
YaccTime_ms	int	Y acceleration time in msec.
YsmoothFactor_ms	int	Y Smooth factor in msec (jerk time).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.2.27 SetBrake

#### Name

**SetBrake**– Set Brake command.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check if the break feature is enabled: (-205)

#### Description

This function sets brake command signal from Inhibit or GPIO connector.

Refer to section [BRAKE] in System.ini configuration file.

Brake commands:

- 1) BrakeCommandSignalState parameter is defined “Direct” in System.ini
  - 0 = Brake OFF
  - 1 = Brake ON
- 2) BrakeCommandSignalState parameter is defined “Inverted” in System.ini
  - 0 = Brake ON
  - 1 = Brake OFF

#### Prototype

```
int SetBrake(  
    int SocketID,  
    int Command  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Command	int	Brake command (0 or 1).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -205: Not enable in your configuration.

### 6.2.2.28 SetGantryMode

#### Name

**SetGantryMode** – Set Gantry mode (Option0, Option1 or Option2).

#### Input tests

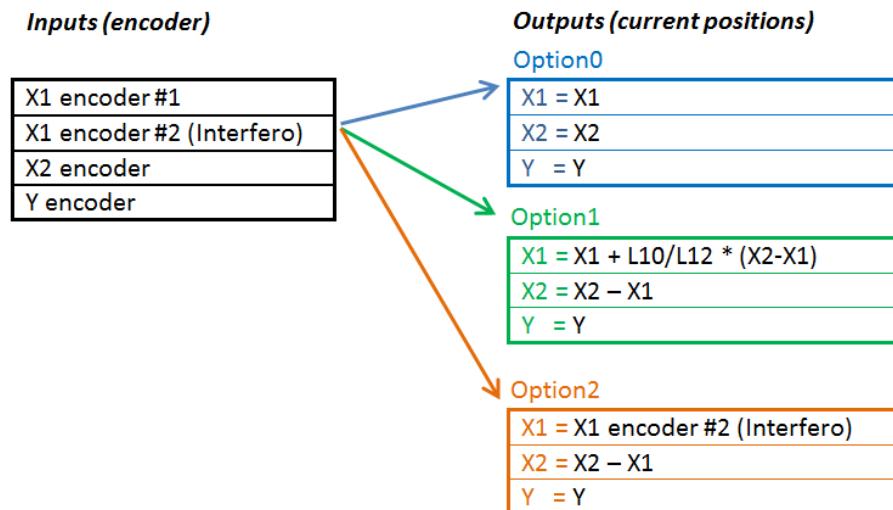
- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the set option: (-17)
- Check the group name is valid: (-19)
- Check if XY gantry mode is enabled: (-205)

#### Description

Set the gantry option to use. It's possible to configure the gantry mode only when the XY group is in “READY” or “DISABLE” state.

Three “gantry” options are available:

- Option0 =>Gantry standard.
- Option1 =>Gantry force balance.
- Option2 =>Gantry force balance with interferometer.



#### Prototype

```
int SetGantryMode(
    int SocketID,
    char * Option
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Option	char *	Option selection(Option0, Option1 or Option2).

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -19: GroupName doesn't exist or unknown command.
- -205: Not enable in your configuration.

### **6.2.2.29 SetJogAcceleration**

#### **Name**

**SetJogAcceleration** – Changes the acceleration parameters in the jog mode.

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group status must be “JOGGING”: (-22)

#### **Description**

This function allows changing the acceleration used by the jog mode. If an error occurs, each positioner stops and the velocity value is set to zero.

To use this function, the jog mode must be enabled (the call of ”StartJog” function is required). If the group status is not “JOGGING” then the (-22) error is returned.

If a slave error or a following error is detected during the jog setting then (-25) error or (-44) error is returned. In this case, the motion is stopped, the jog mode is disabled and the group status becomes “DISABLE”.

#### **Prototype**

```
int SetJogAcceleration(
    int SocketID,
    int XaccelerationTime_ms,
    int XsmoothFactor_ms,
    int YaccelerationTime_ms,
    int YsmoothFactor_ms
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
XaccelerationTime_ms	int	User jog Acceleration time for X in ms.
XsmoothFactor_ms	int	User jog Smooth factor for X in ms.
YaccelerationTime_ms	int	User jog Acceleration time for Y in ms.
YsmoothFactor_ms	int	User jog Smooth factor for Y in ms.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Not allowed due to a positioner error or hardware status.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.30 SetJogVelocity

#### Name

**SetJogVelocity** – Changes “on the fly” the velocity in the jog mode.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group status must be “JOGGING”: (-22)

#### Description

This function allows changing “on the fly” the velocity used by the jog mode in the XY group. If an error occurs, each positioner stops and the velocity value is set to zero.

To use this function, the jog mode must be enabled (the call of the “StartJog” function is required). If the group status is not “JOGGING” then the (-22) error is returned.

If a slave error or a following error is detected during the jog setting then the (-25) error or (-44) error is returned. In this case, the motion is stopped, the jog mode is disabled and the group status becomes “DISABLE”.

#### NOTE

**This function is available only for an XY group.**

#### Prototype

```
int SetJogVelocity(
    int SocketID,
    double Xvelocity,
    double Yvelocity,
    int joystickAckTimeout_ms
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Xvelocity	double	user jog velocity for X in $\mu\text{m/s}$ .
Yvelocity	double	user jog velocity for Y in $\mu\text{m/s}$ .
joystickAckTimeout_ms	int	user jog velocity acknowledge timeout in ms.

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Not allowed due to a positioner error or hardware status.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.31 SetPiston

#### Name

**SetPiston** – Set command to activate piston “Engaged” or “Released”.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check if the brake feature is enabled: (-205)

#### Description

Send command to activate piston “Engaged” or “Released”.

Refer to section [PISTON] in System.ini configuration file.

#### **Piston commands:**

1. PistonCommandSignalState parameter is defined “Direct” in System.ini
  - 0 = Released
  - 1 = Engaged
2. PistonCommandSignalState parameter is defined “Inverted” in System.ini
  - 0 = Engaged
  - 1 = Released

#### Prototype

```
int SetPiston(
```

```
    int SocketID,  
    int Command  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Command	int	Piston command (0 or 1).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -205: Not enable in your configuration.

### 6.2.2.32 SetVarX

#### Name

**SetVarX** – Set new value for a specified parameter name from stages.ini for X positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter value: (-17)
- Check positioner name: (-18)
- Verify the positioner is from a XY group: (-22)

#### Description

Set new value for the specified parameter name from stages.ini file for the X positioner.

#### Prototype

```
int SetVarX(
```

```
    int SocketID,  
    char * ParameterName,  
    double ParameterValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ParameterName	char *	Stages.ini parameter name.
ParameterValue	double	The value to be set for the “ParameterName”.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.33 SetVarXSecondary

#### Name

**SetVarXSecondary** – Set new value for a specified parameter name from stages.ini for X secondary positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter value: (-17)
- Check positioner name: (-18)
- Verify the positioner is from a XY group: (-22)

#### Description

Set new value for the specified parameter name from stages.ini file for the X secondary positioner.

#### Prototype

```
int SetVarXSecondary(  
    int SocketID,  
    char * ParameterName,  
    double ParameterValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ParameterName	char *	Stages.ini parameter name.
ParameterValue	double	The value to be set for the “ParameterName”.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.34 SetVarY

#### Name

**SetVarY** – Set new value for a specified parameter name from stages.ini for Y positioner.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter value: (-17)
- Check positioner name: (-18)
- Verify the positioner is from a XY group: (-22)

#### Description

Set new value for the specified parameter name from stages.ini file for the X secondary positioner.

#### Prototype

```
int SetVarY(  
    int SocketID,  
    char * ParameterName,  
    double ParameterValue  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ParameterName	char *	Stages.ini parameter name.
ParameterValue	double	The value to be set for the “ParameterName”.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.

### 6.2.2.35 SetVelParams

#### Name

**SetVelParams** – Set acceleration parameters for X and Y axes.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check motion profile: (-8)

#### Description

This API updates the current velocity parameters in  $\mu\text{m/s}$  for X and Y axes.

#### Prototype

```
int SetAccParams(
```

```
    int SocketID,  
    double Xvelocity,  
    double Yvelocity  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Xvelocity	double	X velocity in $\mu\text{m/s}$ .
Yvelocity	double	Y velocity in $\mu\text{m/s}$ .

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.2.36 SetXVelParams

#### Name

**SetXVelParams** – Set velocity for X axis.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check motion profile: (-8)

#### Description

This API updates the velocity parameter in  $\mu\text{m/s}$  for X axis.

#### Prototype

```
int SetXVelParams(  
    int SocketID,  
    double Xvelocity  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Xvelocity	double	X velocity in $\mu\text{m/s}$ .

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.2.37 SetYVelParams

#### Name

**SetYVelParams** – Set velocity for Y axis.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check motion profile: (-8)

#### Description

This API updates the velocity parameter in  $\mu\text{m/s}$  for Y axis.

#### Prototype

```
int SetYVelParams(  
    int SocketID,  
    double Yvelocity  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Yvelocity	double	Y velocity in $\mu\text{m/s}$ .

#### Output parameters

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.2.38 SetZone

#### Name

**SetZone** – Set parameters to define the circle zone.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the positioner name is valid: (-18)
- Not allowed due to configuration disabled: (-121)
- Not allowed because group is not initialized or not referenced: (-135)

#### Description

Set parameters to define a circle zone.

An output signal will be generated when the position is inside the circle.

The digital output is defined in the stage.ini file.

It's always available after the home search.

All parameters are defined in  $\mu\text{m}$ .

The Radius and Hysteresis are defined to signal when going out the circle.

#### Prototype

```
int SetZone (int SocketID, double x_center_um, double y_center_um, double  
radius_um, double hysteresis_um)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
x_center_um	double	X center in $\mu\text{m}$ .
y_center_um	double	Y center in $\mu\text{m}$ .
radius_um	double	Radius in $\mu\text{s}$ .
hysteresis_um	double	Hysteresis in $\mu\text{m}$ .

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -18: PositionerName doesn't exist or unknown command.
- -121: Not allowed due to configuration disabled.
- -135: Not allowed because group is not initialized or not referenced.

### 6.2.2.39 StartJog

#### Name

**StartJog** – Enables the jog mode in XY group.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Check the positioner name: (-18)
- Check the group name: (-19)
- Group status must be “READY”: (-22)
- Backlash must be not activated: (-46)

#### Description

This function enables the Jog mode for the first declared XY group. To use this function, the group must be in the “READY” state and all positioners must be idle (means velocity must be 0).

This function allows going to the “JOGGING” state. If the group state is not “READY”, the (-22) error is returned.

#### NOTE

**Use the “EndJog” function to disable the jog mode.**

#### Prototype

```
int GetJogAcceleration(
    int SocketID
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Not allowed due to a positioner error or hardware status.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -46: Not allowed action due to backlash.

### 6.2.2.40 **WaitMotionEnd**

#### **Name**

**WaitMotionEnd** – Wait the end of move (XY group only).

#### **Input tests**

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Check expected position after motion: (-211)

#### **Description**

This function allows waiting the real end of move (after a GroupMoveAbsolute, GroupMoveRelative or GroupMoveSlice).

---

#### **NOTE**

**This function is available only for an XY group.**

---

#### **Prototype**

```
int WaitMotionEnd(
    int SocketID,
    char *GroupName,
    double TimeOutMs,
    double YPosition,
    double XPosition
)
```

#### **Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
GroupName	char *	XY Group name.
TimeOutMs	double	Time out in milliseconds.
YPosition	double	Y position to check in microns.
XPosition	double	X position to check in microns.

#### **Output parameters**

None.

#### **Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -22: Not allowed action.
- -211: Not expected position after motion.

### 6.2.2.41 ZygoADCDiagnosticStatusGet

#### Name

**ZygoADCDiagnosticStatusGet** – Get the diagnostic ADC status.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Axis number and Zygo axis number: (-17)

#### Description

Returns the diagnostic ADC status in relation to the ADC Mux number.

#### Prototype

```
int ZygoADCDiagnosticStatusGet(  
    int SocketID,  
    int Axis,  
    int ADCMuxNumber,  
    char * ADCTDiagStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Axis	int	Zygo axis number (1 or 2).
ADCMuxNumber	int	ADC Mux (refer to table 4-12 from ZMI2402 manual).

#### Output parameters

ADCTDiagStatus      char \*      Raw value from Diag ADC Register.

#### Return (In addition to the results of testing the inputs of functions)

- 0:      No error.
- -2:      TCP timeout.
- -17:      Parameter out of range or incorrect.
- -1000:      Zygo command execution failed.
- -1001:      The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.42 ZygoAmplitudeGet

#### Name

**ZygoAmplitudeGet** – Get the reference channel status and measuring channel amplitude.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Returns the reference channel status and the measuring channel amplitudes.

#### Prototype

```
int ZygoAmplitudeGet(  
    int SocketID,  
    int * ZygoreferenceSignalStatus,  
    int * Meas1Signal,  
    int * Meas2Signal  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

EthernetCommunicationStatus	int *	1 = Connected, 0 = Not connected.
Meas1Signal	int *	Measure 1 Signal Strength (refer to ZMI 2402 Diagnostic ADC).
Meas2Signal	int *	Measure 2 Signal Strength (refer to ZMI 2402 Diagnostic ADC).

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -1001: The controller is not connected to Zygo TCP server. Run  
ZygoStartInterferometer API.

### 6.2.2.43 ZygoConnectToServer

#### Name

**ZygoConnectToServer** – Connect ZYGO to server.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)
- Check Zygo is enabled: (-205)

#### Description

This function opens a TCP/IP communication with a ZMI Measuring board.

#### Prototype

```
int ZygoConnectToServer(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.
- -205: Not enable in your configuration.
- -1002: Connection to Zygo TCP server failed.
- -1003: The XPS controller is already connected to Zygo TCP server.

### 6.2.2.44 ZygoDisconnectFromServer

#### Name

**ZygoDisconnectFromServer** – Disconnect ZYGO from server.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Close the TCP/IP communication opened with the ZYGO ZMI box.

#### Prototype

```
int ZygoDisconnectFromServer(  
    int SocketID  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.2.45 ZygoErrorStatusGet

#### Name

**ZygoErrorStatusGet** – Returns the ZYGO error status code via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Zygo axis value: (-17)
- Check Zygo is enabled: (-205)

#### Description

This function returns the ZYGO axis error code.

The axis error codes are listed in the “Zygo error status list”.

The description of the axis error code can be getting with the “ZygoErrorStatusStringGet” function.

#### Prototype

```
int ZygoErrorStatusGet(  
    int SocketID,  
    int Axis,  
    char * ErrorStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Axis	int	ZMI Axis number (1 = Y axis and 2 = X axis).

#### Output parameters

ErrorStatus	char *	ZMI Axis Error Status.
-------------	--------	------------------------

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -17: Parameter out of range or incorrect.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.46 ZygoErrorStatusStringGet

#### Name

**ZygoErrorStatusStringGet** – Returns the ZYGO axis error status description via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Zygo axis value: (-17)

#### Description

This function returns the ZYGO axis error status description.

#### Prototype

```
int ZygoErrorStatusStringGet(  
    int SocketID,  
    int Axis,  
    char * ErrorStatusDescription  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Axis	int	ZMI Axis number (1 = Y axis and 2 = X axis).

#### Output parameters

ErrorStatusDescription	char *	ZMI Axis Error Status description.
------------------------	--------	------------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.2.47 ZygoEthernetCommunicationStatusGet

#### Name

**ZygoEthernetCommunicationStatusGet** – Get ZYGO Ethernet Connection Status.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Get the Ethernet Communication Status: Connected or Not connected.

#### Prototype

```
int ZygoEthernetCommunicationStatusGet(  
    int SocketID,  
    int * EthernetCommunicationStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

EthernetCommunicationStatus int \* 1 = Connected, 0 = Not connected.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.2.48 ZygoGetPEGLastCommunicationTime

#### Name

**ZygoGetPEGLastCommunicationTime** – Get the last communication time to configure Zygo PEG.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the XPS controller configuration: (-4)

#### Description

This function gets the last communication time to configure Zygo PEG in seconds.

#### Prototype

```
int ZygoGetPEGLastCommunicationTime(  
    int SocketID,  
    double * LastCommunicationTime,  
    )
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
ParamName	double *	Last communication time to configure PEG (s).

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -4: Unknown command.

### 6.2.2.49 ZygoGetVerInterfero

#### Name

**ZygoGetVerInterfero** – Returns firmware version of ZMI system (ZYGO interferometer) via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Zygo is enabled: (-205)

#### Description

This function returns the ZMI system name and its firmware version number.

#### Prototype

```
int ZygoGetVerInterfero(  
    int SocketID,  
    char * Version  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

Version	char *	ZMI system version.
---------	--------	---------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.

### 6.2.2.50 ZygoInterferometerStatusGet

#### Name

**ZygoInterferometerStatusGet** – Get ZYGO interferometer status.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Get ZYGO interferometer status:

- Ethernet connection status.
- Axis #1 signal status.
- Axis #2 signal status.
- Reference signal status.
- P2 Board status.

#### Prototype

```
int ZygoInterferometerStatusGet(  
    int SocketID,  
    int * EthernetCommunicationStatus,  
    int * ZygoAxis1MeasureSignal,  
    int * ZygoAxis2MeasureSignal,  
    int * ZygoReferenceSignalStatus,  
    int * ZygoP2BoardStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

EthernetCommunicationStatus	int *	1 = Connected, 0 = Not connected.
ZygoAxis1MeasureSignal	int *	1 = Axis #1 signal present, 0 = No signal.
ZygoAxis2MeasureSignal	int *	1 = Axis #2 signal present, 0 = No signal.
ZygoReferenceSignalStatus	int *	1 = Reference signal present, 0 = No Signal.
ZygoP2BoardStatus	int *	1 = P2 board ready, 0 = P2 board not ready or not present.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.

### 6.2.2.51 ZygoPositionGet

#### Name

**ZygoPositionGet** – Returns positions of Y and X axes from ZMI system via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Zygo is enabled: (-205)

#### Description

This function returns positions (36-bits) of Y and X axes.

#### Prototype

```
int ZygoPositionGet(  
    int SocketID,  
    long long int * PositionY,  
    long long int * PositionX  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

PositionY	long long int *	Current position of Y axis.
PositionX	long long int *	Current position of X axis.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.52 ZygoReadLong

#### Name

**ZygoReadLong** – Read a long register from a ZMI Measuring board.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function reads a LONG register (max 32-bits) from a ZMI Measuring board.

#### Example

*Read Control Register 1 on Axis 1:*

**ZygoReadLong (AXIS1, A)**

*ZMI Measuring board Response:*

#H70

#### Prototype

```
int ZygoReadLong(  
    int SocketID,  
    char * AxisNum,  
    char * Register,  
    char * Response  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
AxisNum	char *	“AXIS1” or “AXIS2”.
Register	char *	Register value in hexadecimal.

#### Output parameters

Response	char *	Response returned by the ZMI Measuring board.
----------	--------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.53 ZygoReadWord

#### Name

**ZygoReadWord** – Read a long register from a ZMI Measuring board.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function reads a WORD register data (max 16-bits) from a ZMI Measuring board.

#### Example

*Read Control Register 1 on Axis 2:*

**ZygoReadWord (AXIS2, A)**

*ZMI Measuring board Response:*

#H1

#### Prototype

```
int ZygoReadWord(  
    int SocketID,  
    char * AxisNum,  
    char * Register,  
    char * Response  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
AxisNum	char *	“AXIS1” or “AXIS2”.
Register	char *	Register value in hexadecimal.

#### Output parameters

Response	char *	Response returned by the ZMI Measuring board.
----------	--------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.54 ZygoRegisterGet

#### Name

**ZygoRegisterGet** – Get register value from P2 ZYGO board.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)

#### Description

This function reads the register value from P2 ZYGO board.

Refer to the list of P2 interface registers.

#### Prototype

```
int ZygoRegisterGet(  
    int SocketID,  
    char * PositionerName,  
    int Register,  
    int * Value  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
Register	int	Register value in hexadecimal.

#### Output parameters

Value	int *	Response returned by the ZMI Measuring board.
-------	-------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.2.55 ZygoRegisterSet

#### Name

**ZygoRegisterSet** – Set register value from P2 ZYGO board.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check the object type of this command is valid: (-8)

#### Description

This function sets the register value from P2 ZYGO board.

Refer to the list of P2 interface registers.

#### Prototype

```
int ZygoRegisterSet(
```

```
    int SocketID,  
    char * PositionerName,  
    int Register,  
    int Value  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
PositionerName	char *	Positioner name.
Register	int	Register value in hexadecimal.
Value	int	Response returned by the ZMI Measuring board.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -8: Wrong object type for this command.

### 6.2.2.56 ZygoReset

#### Name

**ZygoReset** – Reset all Zygo axes.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check if Zygo mode is enabled: (-205)
- Check Zygo TCP Server Connexion: (-1001)

#### Description

This function sends commands to Zygo ZMI 2402 to reset all Zygo axes (#1 and #2).

#### Prototype

```
int ZygoReset(  
    int SocketID  
);
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -205: Not enable in your configuration.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.

### 6.2.2.57 ZygoResetX

#### Name

**ZygoResetX** – Reset X axis via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Zygo is enabled: (-205)

#### Description

This function resets X axis (defined as Zygo axis #2).

The **Axis Reset** resets only the measurement function and error conditions, and does not affect axis configuration. An axis reset is identical to the combined effects of the **Position Reset**, **Time Reset**, and **Error Reset**.

The **Position Reset** reinitializes the position measurement function and sets the position register to zero.

The **Time Reset** immediately sets the time register to zero.

The **Error Reset** resets all errors.

#### **NOTE**

**Due to minor timing differences, simultaneous multi-axis reset commands may result in some axes actually being reset one reference clock period (50 ns) later than other axes.**

The allowable operating conditions and the time required for an axis reset depends on the settings in *Aux Control 0* as shown below.

The **reset time** is from the *Reset Axis* command until *Position Reset Complete* status.

This time includes the *Reset Delay* (RD) specified in *Control Register 2*. Table 3-8 specifies the *Reset Delay* choices.

#### Prototype

```
int ZygoResetX(
    int SocketID
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.

### 6.2.2.58 ZygoResetY

#### Name

**ZygoResetY** – Reset Y axis via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Zygo is enabled: (-205)

#### Description

This function resets Y axis (defined as Zygo axis #1).

The **Axis Reset** resets only the measurement function and error conditions, and does not affect axis configuration. An axis reset is identical to the combined effects of the **Position Reset**, **Time Reset**, and **Error Reset**.

The **Position Reset** reinitializes the position measurement function and sets the position register to zero.

The **Time Reset** immediately sets the time register to zero.

The **Error Reset** resets all errors.

#### NOTE

**Due to minor timing differences, simultaneous multi-axis reset commands may result in some axes actually being reset one reference clock period (50 ns) later than other axes.**

The allowable operating conditions and the time required for an axis reset depends on the settings in *Aux Control 0* as shown below.

**Table 3-7 Axis Reset Conditions and Required Time**

<u>Enable Reset</u> <u>Finds Velocity</u> <u>Aux Control 0</u>	<u>RFV Mode</u> <u>Aux Control 0</u>	Reset Time	Max Velocity during reset	Max Acceleration during reset
0	-	22 µs + RD	±0.1 m/s	10 g
1	0 (Diagnostic)	2.0 ms + RD	±2.1 m/s	0.1 g
1	1 (Default)	210 µs + RD	±2.1 m/s	10 g

The **reset time** is from the *Reset Axis* command until *Position Reset Complete* status.

This time includes the *Reset Delay* (RD) specified in *Control Register 2*. Table 3-8 specifies the *Reset Delay* choices.

**Table 3-8 Reset Delay**

RD2	RD1	RD0	Reset Delay (RD)
0	0	0	2 µs
0	0	1	6 µs
0	1	0	26 µs
0	1	1	102 µs
1	0	0	410 µs
1	0	1	1.6 ms
1	1	0	6.5 ms
1	1	1	26.2 ms

**Prototype**

```
int ZygоБезетY(  
    int SocketID  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
----------	-----	--

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -205: Not enable in your configuration.
- -1000: Zygо command execution failed.
- -1001: The controller is not connected to Zygо TCP server. Run ZygоБезетInterferometer API.

### 6.2.2.59 ZygoSendAndReceive

#### Name

**ZygoSendAndReceive** – Send a command and read a response from ZYGO box.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Send a command to ZYGO box and wait a response from ZYGO box.

The command string must be defined in the ZYGO format.

#### Prototype

```
int ZygoSendAndReceive(  
    int SocketID,  
    char * Command,  
    char * Response  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Command	char *	“AXIS1” or “AXIS2”.
Register	char *	Command to send to ZYGO box.

#### Output parameters

Response	char *	Response read from ZYGO box.
----------	--------	------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.60 ZygoSetOffsetX

#### Name

**ZygoSetOffsetX** – Set offset value for X axis via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter type: (-127)
- Check Zygo is enabled: (-205)

#### Description

This function sets the offset value for X (defined as Zygo axis #2).

The offset value is subtracted from the position value.

#### Prototype

```
int ZygoSetOffsetX(  
    int SocketID,  
    long Xoffset  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Xoffset	long	Offset value for X.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -127: Wrong parameter type in the command string: long or long \* expected.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.

### 6.2.2.61 ZygoSetOffsetY

#### Name

**ZygoSetOffsetY** – Set offset value for Y axis via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter type: (-127)
- Check Zygo is enabled: (-205)

#### Description

This function sets the offset value for Y (defined as Zygo axis #1).

The offset value is subtracted from the position value.

#### Prototype

```
int ZygoSetOffsetY(
```

```
    int SocketID,  
    long Yoffset  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Yoffset	long	Offset value for Y.

#### Output parameters

None.

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -127: Wrong parameter type in the command string: long or long \* expected.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.

### 6.2.2.62 ZygoSetPEGParams

#### Name

**ZygoSetPEGParams** – Set PEG parameters (ZYGO System) via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameter type: (-127)
- Check Zygo is enabled: (-205)

#### Description

This function sets PEG (Position Event Generator) parameters.

The PEG function generates an electronic signal when the internal position measurement meets criteria established by values programmed into the board by the host system.

The first pulse is output when the position value enters the active region between the P1 and P2 values.

Additional pulses are output each time the position changes by the prescribed *Delta 1*, or in *Delta 2* mode

For K1 number of PEG events, the increment value is *Delta 1*

For K2 number of PEG events, the increment value is *Delta 2*

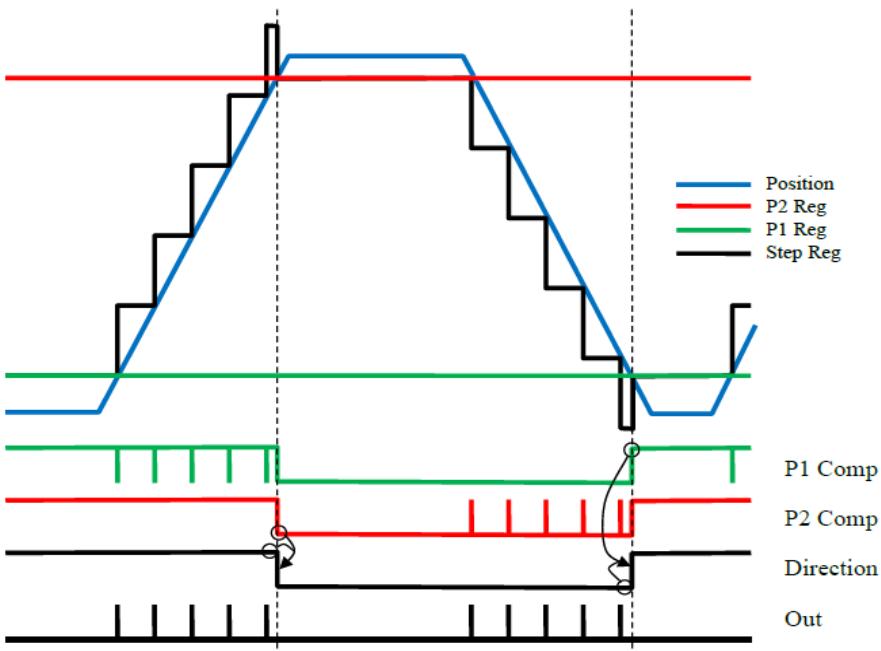
This repeats K1, K2, K1, K2, etc...

Pulses stop when the position leaves the active region between the P1 and P2 values.

This may be repeated for travel in either direction between P1 and P2.

A number of other features, such as signal polarity, axis select (for 2402), and pulse duration shall be programmable. Direction of motion is inferred by the position crossing the start and stop values. Starting operation at a point in between the start and stop values is undefined.

When entering at P1, the PEG starts with Delta 1. When entering at P2, the Delta is selectable by the *PEG P2 Delta* bit in the PEG Control register (refer to PEG Control register).



---

**NOTE**

**This API is enabled only with a ZMI system equipped of PEG function.**

**Refer to ZYGO ZMI 2400 with PEG Function (revision J).**

---

**Prototype**

```
int ZygoSetPEGParams(
    int SocketID,
    long P1,
    long P2,
    unsigned int Delta1,
    long K1,
    unsigned int Delta2,
    long K2,
    int ControlWord
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
P1	long	Start PEG region (+/- 660 mm).
P2	long	End PEG region (+/- 660 mm and P2 >P1).
Delta1	unsigned int	Increment value (max 20 µm).
K1	long	Number of PEG events.
Delta2	unsigned int	Increment value (max 20 µm).
K2	long	Number of PEG events.
ControlWord	int	PEG control (refer to PEG Control register).

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -127: Wrong parameter type in the command string: long or long \* expected.
- -205: Not enable in your configuration.
- -1000: Zygote command execution failed.
- -1001: The controller is not connected to Zygote TCP server. Run ZygoteStartInterferometer API.

### 6.2.2.63 ZygoStartBoardP2

#### Name

**ZygoStartBoardP2**– Starts ZYGO P2 board.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check input parameters value: (-17)

#### Description

This function allows initializing and starting the ZYGO P2 board.

#### ***Required settings to get the board up and running after power up:***

##### Switch Settings

- |         |   |
|---------|---|
| SW6-3:4 | Set to ON ON for INT0 to deliver Axis 1 interrupts            |
| SW6-5:6 | Set to OFF ON for INT1 to deliver Axis 2 interrupts           |
| SW6-7:8 | Set to OFF OFF for sampling on SCLK1 falling edge             |
| SW8-1:8 | Set all to ON for a 00000000 base P2 address (recommendation) |

This will correspond to required A11:4 on the P2 pins.

- Switch settings can be found on **2-9** in the manual.

##### Register Settings

Control Register 1 Set the desired P2\_RESET \* line behavior (bits 6:3)

Bit 8 (P2 latch Mode) = 0, External sample input

Bit 9 (P2 D16 Select) = 0, for 32-bit data path

- The above must be done for both axes.

Bit 10 (P2 Sclk0 Disable) = 1

Bit 11 (P2 Sclk0 Output Enable) = 1

- The above must be done for axis 1 only.

Control Register 2 Set the desired  $K_p$  digital filter (bits 6:4)

the desired  $K_v$  digital filter values, (bits 2:0)

- See section of ZMI manual for more information

Set the *Change Position Direction Sense* bit. This bit defines the interferometer direction.

Setting this bit reverses the direction sense of the position accumulator. An axis reset must be performed after changing the direction sense.

- All of the above must be done for both axes.

Data Age Adjust Program the appropriate data age adjust values

See detailed procedure below for more information

- This must be done for both axes.

P2 Interrupt Enable Select which errors will generate a signal on the P2\_INT \*

- This must be done for both axes.

#### Procedural Steps

Read the status register until *Reference Present* bit is present in the *Status Register*. This indicates that the laser is locked. Check the *Measure Present* bit on each axis; when this bit is active, perform an axis reset for that axis. Ensure Error Status Register is cleared. Position information should now be continuously updated, as long as there are no Fatal Errors.

**Prototype**

```
int ZygoStartBoardP2(  
    int SocketID,  
    int Kv,  
    int Kp,  
    bool ReverseDirectionSense,  
    int DataAgeAdjust  
)
```

**Input parameters**

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Kv	int	digital filter Kv gain = -7, -9, -11, -13, -15, -17, -19 or -32.
Kp	int	digital filter Kp gain = -2, -3,-4, -5, -6, -7, -8 or -9.
ReverseDirectionSense	bool	true = reversed direction, false = normal direction.
DataAgeAdjust	int	Data Age Adjust Register.

**Output parameters**

None.

**Return** (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -1001: The controller is not connected to Zygote TCP server. Run ZygoStartInterferometer API.

### 6.2.2.64 ZygoStartInterferometer

#### Name

**ZygoStartInterferometer** – Start ZYGO interferometer.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

Start ZYGO interferometer: Open Ethernet connection, Check signals, Start P2 Board  
This API can take several minutes.

#### Prototype

```
int ZygoADCDiagnosticStatusGet(  
    int SocketID,  
    int Axis,  
    int ADCMuxNumber,  
    char * ADCCDiagStatus  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Axis	int	Zygo axis number (1 or 2).
ADCMuxNumber	int	ADC Mux (refer to table 4-12 from ZMI2402 manual).

#### Output parameters

ADCCDiagStatus      char \*      Raw value from Diag ADC Register.

#### Return (In addition to the results of testing the inputs of functions)

- 0:      No error.
- -2:      TCP timeout.
- -17:      Parameter out of range or incorrect.
- -205:      Not enable in your configuration.
- -1000:      Zygo command execution failed.
- -1001:      The controller is not connected to Zygo TCP server. Run. ZygoStartInterferometer API.
- -1002:      Connection to Zygo TCP server failed.
- -1003:      The XPS controller is already connected to Zygo TCP server.
- -1004:      Zygo signal is not present.

### 6.2.2.65 ZygoStatusGet

#### Name

**ZygoStatusGet** – Returns the ZYGO axis status code via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Zygo axis value: (-17)
- Check Zygo is enabled: (-205)

#### Description

This function returns the ZYGO axis status code.

The axis status codes are listed in the “Zygo axis status list”.

The description of the axis status code can be getting with the “ZygoStatusStringGet” function.

#### Prototype

```
int ZygoErrorStatusGet(  
    int SocketID,  
    int Axis, char * Status  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Axis	int	ZMI Axis number (1 = Y axis and 2 = X axis).

#### Output parameters

Status	char *	ZMI Axis Status.
--------	--------	------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -17: Parameter out of range or incorrect.
- -205: Not enable in your configuration.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.66 ZygoStatusStringGet

#### Name

**ZygoStatusStringGet** – Returns the ZYGO axis status description via Ethernet.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.
- Check Zygo axis value: (-17)

#### Description

This function returns the ZYGO axis status description.

#### Prototype

```
int ZygoStatusStringGet(  
    int SocketID,  
    int Axis,  
    char * StatusDescription  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
Axis	int	ZMI Axis number (1 = Y axis and 2 = X axis).

#### Output parameters

StatusDescription	char *	ZMI Axis Status description.
-------------------	--------	------------------------------

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -17: Parameter out of range or incorrect.

### 6.2.2.67 ZygoWriteLong

#### Name

**ZygoWriteLong** – Write data to a long register of ZMI Measuring board.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function write data to a long register (max 32-bits) of ZMI Measuring board.

#### Example

*Write 0x5000 to Control Register 1 on Axis 1:*

**ZygoWriteLong(AXIS1, A, 5000)**

*ZMI Measuring board Response:*

**OK**

#### Prototype

```
int ZygoWriteLong(  
    int SocketID,  
    char * AxisNum,  
    char * Register,  
    char * Data,  
    char * Response  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
AxisNum	char *	“AXIS1” or “AXIS2”.
Register	char *	Register value in hexadecimal.
Data	char *	Data value in hexadecimal in max-32bits.

#### Output parameters

Response	char *	Response returned by the ZMI Measuring board.
----------	--------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

### 6.2.2.68 ZygoWriteWord

#### Name

**ZygoWriteWord** – Write data to a word register of ZMI Measuring board.

#### Input tests

- Refer to section 6.1: “Input Tests Common to all XPS Functions”.

#### Description

This function changes a word register data (max 16-bits) to a ZMI Measuring board.

#### Example

*Write 0x0020 to Control Register 1 on Axis 1:*

**ZygoWriteWord(AXIS1, A, 20)**

*ZMI Measuring board Response:*

**OK**

#### Prototype

```
int ZygoWriteWord(  
    int SocketID,  
    char * AxisNum,  
    char * Register,  
    char * Data,  
    char * Response  
)
```

#### Input parameters

SocketID	int	Socket identifier gets by the “TCP_ConnectToServer” function.
AxisNum	char *	“AXIS1” or “AXIS2”.
Register	char *	Register value in hexadecimal.
Data	char *	Data value in hexadecimal in max-16bits.

#### Output parameters

Response	char *	Response returned by the ZMI Measuring board.
----------	--------	---

#### Return (In addition to the results of testing the inputs of functions)

- 0: No error.
- -2: TCP timeout.
- -1000: Zygo command execution failed.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
- Zygo errors: Refer to section 7.9: “Error List”.

**7.****Lists and Tables for XPS Functions****7.1 Event Triggers List**

[Actor.]			[Category.]				Event Name	Parameters					
Group	Positioner	GPIO	TimerX	SGamma	Log	XYLineArc	Spline	PVT	PT	1	2	3	4
										Immediate	0	0	0
										Always	0	0	0
			■							Timer	0	0	0
■			■ ■							MotionStart	0	0	0
■			■ ■							MotionEnd	0	0	0
■			■ ■							MotionState	0	0	0
■			■							MotionDone	0	0	0
■			■							MotionDoneTimeout	0	0	0
■			■ ■							ConstantVelocityStart	0	0	0
■			■							ConstantVelocityEnd	0	0	0
■			■ ■							ConstantVelocityState	0	0	0
■			■							ConstantAccelerationStart	0	0	0
■			■							ConstantAccelerationEnd	0	0	0
■			■							ConstantAccelerationState	0	0	0
■			■							ConstantDecelerationStart	0	0	0
■			■							ConstantDecelerationEnd	0	0	0
■			■							ConstantDecelerationState	0	0	0
■				■ ■ ■ ■						TrajectoryStart	0	0	0
■				■ ■ ■ ■						TrajectoryEnd	0	0	0
■				■ ■ ■ ■						TrajectoryState	0	0	0
■				■ ■ ■ ■						ElementNumberStart	Element index	0	0
■				■ ■ ■ ■						ElementNumberState	Element index	0	0
■				■ ■ ■ ■						TrajectoryPulse	0	0	0
■				■ ■ ■ ■						TrajectoryPulseState	0	0	0
	■									DILowState	Bit index	0	0
	■									DIHighState	Bit index	0	0
	■									DILowHigh	Bit index	0	0
	■									DIHighLow	Bit index	0	0
	■									DToggled	Bit index	0	0
	■									ADCHighLimit	Value	0	0
	■									ADCLowLimit	Value	0	0
	■									ADCInWindow	min	max	0
	■									ADCOutWindow	min	max	0
■										PositionerError	Mask	0	0
■										PositionerHardwareStatus	Mask	0	0
■										ExcitationSignalStart	0	0	0
■										ExcitationSignalEnd	0	0	0
■										WarningFollowingError	0	0	0
■										WaitForPositionLeftToRight	Target position	0	0
■										WaitForPositionRightToLeft	Target position	0	0
■										WaitForPositionToggled	Target position	0	0
										DoubleGlobalArrayEqual	Global variable number	value	0
										DoubleGlobalArrayDifferent	Global variable number	value	0
										DoubleGlobalArrayInferiorOrEqual	Global variable number	value	0
										DoubleGlobalArraySuperiorOrEqual	Global variable number	value	0

									DoubleGlobalArrayInferior	Global variable number	value	0	0
									DoubleGlobalArraySuperior	Global variable number	value	0	0
									DoubleGlobalArrayInWindow	Global variable number	min	max	0
									DoubleGlobalArrayOutWindow	Global variable number	min	max	0

## 7.2 Actions List

[Actor.]			Action Name	Parameters			
Group	Positioner	GPIO	TimerX	1	2	3	4
	■		DACSet.CurrentPosition	Positioner name	Gain	Offset	0
	■		DACSet. CurrentVelocity	Positioner name	Gain	Offset	0
	■		DACSet. SetpointPosition	Positioner name	Gain	Offset	0
	■		DACSet. SetpointVelocity	Positioner name	Gain	Offset	0
	■		DACSet. SetpointAcceleration	Positioner name	Gain	Offset	0
	■		DACSet.Value	Value	0	0	0
	■		DOPulse	Mask	0	0	0
	■		DOToggle	Mask	0	0	0
	■		DOSet	Mask	Value	0	0
			EventRemove	Trigger identifier (-1 for itself)	0	0	0
			ExecuteCommand	Function name	Parameters (Between {} and separator is the semi-column)	Task name	
			ExecuteTCLScript	TCL file name	Task name	Arguments	
			ExternalGatheringRun	Nb of points	Divisor	0	0
			GatheringOneData	0	0	0	0
			GatheringRun	Nb of points	Divisor	0	0
			GatheringRunAppend	0	0	0	0
			GatheringStop	0	0	0	0
			GlobalArrayDoubleSet	Global variable number	Double value	0	0
			GlobalArrayStringSet	Global variable number	String value	0	0
			KillTCLScript	Task name	0	0	0
■			MoveAbort	0	0	0	0
■			MoveAbortFast	Deceleration multiplier	0	0	0
			SynchronizeProfiler	0	0	0	0

### 7.3 Gathering Data Types

*PositionerName.CurrentAcceleration*  
*PositionerName.CurrentPosition*  
*PositionerName.CurrentVelocity*  
*PositionerName.SetpointAcceleration*  
*PositionerName.SetpointPosition*  
*PositionerName.SetpointVelocity*  
*PositionerName.FollowingError*  
*PositionerName.FollowingErrorCompensation (for precision platform only)*  
*PositionerName.InnerFollowingError (for precision platform only)*  
*PositionerName.ExcitationSignalInput*  
*PositionerName.CorrectedEncoderPosition*  
*PositionerName.CorrectedSetpointPosition*  
*PositionerName.CorrectorOutput*  
*PositionerName.EstimatedVelocity*  
*PositionerName. CorrectorOutput BeforeCompensation (for precision platform only)*  
*PositionerName. CorrectorOutput BeforeCompensationFiltered (precision platform)*  
*PositionerName. CorrectorOutput BeforeDamperFilter (for precision platform only)*  
*PositionerName. CorrectorOutput DamperFilter (for precision platform only)*  
*PositionerName. CorrectorOutputDualPID (for precision platform only)*  
*PositionerName. CorrectorOutputPID (for precision platform only)*  
*PositionerName.RawCorrectorOutput (for precision platform only)*  
*PositionerName.RawCurrentPosition (for precision platform only)*  
*PositionerName.FilteredPIDOutputBeforeFeedForward (for precision platform only)*  
*PositionerName.GantryOption (for precision platform only)*

#### **ISA Hardware:**

- GPIO1.DI
- GPIO1.DO
- GPIO2.DI
- GPIO3.DI
- GPIO3.DO
- GPIO4.DI
- GPIO4.DO
- GPIO2.ADC1
- GPIO2.ADC2
- GPIO2.ADC3
- GPIO2.ADC4
- GPIO2.DAC1
- GPIO2.DAC2
- GPIO2.DAC3
- GPIO2.DAC4

**PCI Hardware:****Basic GPIO board:**

- GPIO1.DI
- GPIO1.DO
- GPIO2.ADC1
- GPIO2.ADC2
- GPIO2.DAC1
- GPIO2.DAC2

**Extended GPIO board:**

- GPIO3.DI
- GPIO3.DO
- GPIO5.DI
- GPIO5.DO
- GPIO6.DI
- GPIO6.DO
- GPIO4.ADC1
- GPIO4.ADC2
- GPIO4.ADC3
- GPIO4.ADC4
- GPIO4.ADC5
- GPIO4.ADC6
- GPIO4.ADC7
- GPIO4.ADC8
- GPIO4.DAC1
- GPIO4.DAC2
- GPIO4.DAC3
- GPIO4.DAC4
- GPIO4.DAC5
- GPIO4.DAC6
- GPIO4.DAC7
- GPIO4.DAC8

ISRCorrectorTimePeriod

ISRCorrectorTimeUsage

ISRProfilerTimeUsage

ISRServitudesTimeUsage

CPUTotalLoadRatio

## 7.4 External Gathering Data Types

*PositionerName.ExternalLatchPosition*

### ISA Hardware:

- GPIO2.ADC1
- GPIO2.ADC2
- GPIO2.ADC3
- GPIO2.ADC4
- GPIO2.DAC1
- GPIO2.DAC2
- GPIO2.DAC3
- GPIO2.DAC4

### PCI Hardware:

#### **Basic GPIO board:**

- GPIO2.ADC1
- GPIO2.ADC2
- GPIO2.DAC1
- GPIO2.DAC2

#### **Extended GPIO board:**

- GPIO4.ADC1
- GPIO4.ADC2
- GPIO4.ADC3
- GPIO4.ADC4
- GPIO4.ADC5
- GPIO4.ADC6
- GPIO4.ADC7
- GPIO4.ADC8
- GPIO4.DAC1
- GPIO4.DAC2
- GPIO4.DAC3
- GPIO4.DAC4
- GPIO4.DAC5
- GPIO4.DAC6
- GPIO4.DAC7
- GPIO4.DAC8

## 7.5 Positioner Error List

Bit #	Mask	Error description
0	1	General inhibition detected
1	2	Fatal following error detected
2	4	Home search time out
3	8	Motion done time out
4	16	Requested position exceed travel limits in trajectory or slave mode
5	32	Requested velocity exceed maximum value in trajectory or slave mode
6	64	Requested acceleration exceed maximum value in trajectory or slave mode
7	168	Clamping incoherence
8	256	Minus end of run activated
9	512	Plus end of run activated
10	1024	Minus end of run glitch
11	2048	Plus end of run glitch
12	4096	Encoder quadrature error
13	8192	Encoder frequency and coherancy error
14	16384	Sine and Cosine radius error
15	32768	X or Y correction is out of encoder correction limits
16	65536	Hard interpolator encoder error
17	131072	Hard interpolator encoder quadrature error
20	1048576	First driver in fault
21	2097152	Second driver in fault
22	4194304	AqB and Sine/Cosine out of phase
24	16777216	Home search mechanical zero inconsistency
25	33554432	Interferometer no signal error on axis or reference
26	67108864	Interferometer glitch error on axis or reference
27	134217728	Fatal internal error
28	268435456	GPIO transfer error
29	536870912	I2C transfer error

---

### NOTE

The most significant bit is always set to 1. So, all positioner errors are negative.

---

## 7.6 Positioner Hardware Status List

Bit #	Mask	Error description
0	1	General inhibition detected
2	4	ZM high level
8	256	Minus end of run activated
9	512	Plus end of run activated
10	1024	Minus end of run glitch
11	2048	Plus end of run glitch
12	4096	Encoder quadrature error
13	8192	Encoder frequency or coherancy error
14	16384	PCI Sine and Cosine radius error
15	32768	X or Y correction is out of encoder correction limits
16	65536	Hard interpolator encoder error
17	131072	Hard interpolator encoder quadrature error
20	1048576	First driver in fault
21	2097152	Second driver in fault
22	4194304	First driver powered on
23	8388608	Second driver powered on
24	16777216	Interferometer no signal error on axis or reference
25	33554432	Interferometer glitch error on axis or reference
26	67108864	PCI PCO error (underrun)
27	134217728	PCI PCO pulses ended
28	268435456	PCI GPIO External gathering error
29	536870912	PCI I2C transfer error
30	1073741824	PCI External gathering error

### NOTE

Positioner errors are used to trigger consequences on the system, for instance disable, emergency break, etc. Positioner hardware status information is mainly provided for information purposes.

#### Positioner hardware status description:

*General inhibition detected:* This refers to the General Inhibition connector at the rear panel or the Stop All button at the front panel of the XPS controller. The General Inhibiton connector is a safety feature and can be used for a custom STOP ALL emergency switch. Inhibition (pin#2), must always be connected to GND during normal operation of the controller. In this case, inhibition is not detected. An open circuit is equivalent to pressing STOP ALL on the front panel, in which case, inhibition is detected.

*ZM high level:* This refers to the mechanical zero signal used with some stages. The ZM signal is high during one part of the travel and low during the other part of the travel. The detection of the ZM high/low transition in combination with an encoder index pulse signal allows a fast and repeatable origin search (MechanicalZeroAndIndexHomeSearch).

*Minus end of run activated:* Refers to the hardware minus end of run limit switch. During normal operation, this end of run switch should never be activated and any motion will be stopped by the detection of the minus software limit.

*Plus end of run activated:* Refers to the hardware positive end of run limit switch. During normal operation, this end of run switch should never be activated and any motion will be stopped by the detection of the positive software limit.

*Minus end of run glitch:* Undesirable, momentary instability of the hardware minus end of run signal, for instance can be generated by ripple or noise.

*Plus end of run glitch:* Undesirable, momentary instability of the hardware positive end of run signal, for instance can be generated by ripple or noise.

*Encoder quadrature error:* Error generated when the signals of both encoder channels simultaneously change. In normal operation, only one quadrature signal changes state at a time. This error can occur due to an undesirable level change or a glitch as illustrated below.

*Encoder freq. and coherency error:* Error generated when the frequency of the signals is too high. The maximum frequency of the encoder input is 25MHz.

*Hard interpolator encoder error:* Error generated when the difference of the sine/cosine encoder signals from a unity circle is too large (for instance when signals are phase shifted or amplitude modified).

*Hard interpolator quad. encoder error:* Error generated when the signals of both encoder channels of the hardware interpolated encoder output simultaneously change. Same error as *Encoder quadrature error* except that the quadrature signals are those converted from the sine/cosine signals of the hard interpolator. The hardware interpolator is used only with AnalogInterpolated encoders to trigger the position compare output and to gather positions during external data gathering.

*First driver in fault:* problem with the first driver.

*Second driver in fault:* problem with the second driver in case two drivers are connected to one axis.

*First driver powered on:* First driver with motor ON after initialization.

*Second driver powered on:* Second driver with motor ON after initialization, in case two drivers are connected to one axis.

## 7.7 Positioner Driver Status List

<b>Bit</b>	<b>code</b>	<b>DRV00x</b>	<b>DRV01</b>	<b>DRV02x</b>	<b>D6U</b>	<b>DRV03</b>	<b>DRVP1</b>
<b>0</b>	<b>a</b>			Short-circuit	Short-circuit	Short-circuit	
<b>1</b>	<b>b</b>			Broken fuse	Broken fuse	Broken fuse	Voltage out of range
<b>2</b>	<b>c</b>			Thermistor fault	Thermistor fault		Over temperature
<b>3</b>	<b>d</b>			Initialization error	Initialization error	Initialization error	Initialization error
<b>4</b>	<b>e</b>			I <sup>2</sup> T	I <sup>2</sup> T	I <sup>2</sup> T	Dynamic error
<b>5</b>	<b>f</b>			Current limit	Current limit		
<b>6</b>	<b>g</b>					TG is opened	No stage connected
<b>7</b>	<b>h</b>	Inhibition input	Inhibition input	Inhibition input	Inhibition input	Inhibition input	Inhibition input
<b>8</b>	<b>i</b>	Driver in fault	Driver in fault	Driver in fault	Driver in fault	Driver in fault	Driver in fault

## 7.8 Group Status List

Code	Description
<b>0</b>	NOTINIT state
<b>1</b>	NOTINIT state due to an emergency brake: see positioner status
<b>2</b>	NOTINIT state due to an emergency stop: see positioner status
<b>3</b>	NOTINIT state due to a following error during homing
<b>4</b>	NOTINIT state due to a following error
<b>5</b>	NOTINIT state due to an homing timeout
<b>6</b>	NOTINIT state due to a motion done timeout during homing
<b>7</b>	NOTINIT state due to a KillAll command
<b>8</b>	NOTINIT state due to an end of run after homing
<b>9</b>	NOTINIT state due to an encoder calibration error
<b>10</b>	Ready state due to an AbortMove command
<b>11</b>	Ready state from homing
<b>12</b>	Ready state from motion
<b>13</b>	Ready State due to a MotionEnable command
<b>14</b>	Ready state from slave
<b>15</b>	Ready state from jogging
<b>16</b>	Ready state from analog tracking
<b>17</b>	Ready state from trajectory
<b>18</b>	Ready state from spinning
<b>19</b>	Ready state due to a group interlock error during motion
<b>20</b>	Disable state
<b>21</b>	Disabled state due to a following error on ready state
<b>22</b>	Disabled state due to a following error during motion
<b>23</b>	Disabled state due to a motion done timeout during moving
<b>24</b>	Disabled state due to a following error on slave state
<b>25</b>	Disabled state due to a following error on jogging state
<b>26</b>	Disabled state due to a following error during trajectory
<b>27</b>	Disabled state due to a motion done timeout during trajectory
<b>28</b>	Disabled state due to a following error during analog tracking
<b>29</b>	Disabled state due to a slave error during motion
<b>30</b>	Disabled state due to a slave error on slave state
<b>31</b>	Disabled state due to a slave error on jogging state
<b>32</b>	Disabled state due to a slave error during trajectory
<b>33</b>	Disabled state due to a slave error during analog tracking
<b>34</b>	Disabled state due to a slave error on ready state
<b>35</b>	Disabled state due to a following error on spinning state
<b>36</b>	Disabled state due to a slave error on spinning state
<b>37</b>	Disabled state due to a following error on auto-tuning
<b>38</b>	Disabled state due to a slave error on auto-tuning
<b>39</b>	Disable state due to an emergency stop on auto-tuning state
<b>40</b>	Emergency braking
<b>41</b>	Motor initialization state
<b>42</b>	Not referenced state
<b>43</b>	Homing state
<b>44</b>	Moving state

<b>45</b>	Trajectory state
<b>46</b>	Slave state due to a SlaveEnable command
<b>47</b>	Jogging state due to a JogEnable command
<b>48</b>	Analog tracking state due to a TrackingEnable command
<b>49</b>	Analog interpolated encoder calibrating state
<b>50</b>	NOTINIT state due to a mechanical zero inconsistency during homing
<b>51</b>	Spinning state due to a SpinParametersSet command
<b>52</b>	NOTINIT state due to a clamping timeout
<b>55</b>	Clamped
<b>56</b>	Ready state from clamped
<b>58</b>	Disabled state due to a following error during clamped
<b>59</b>	Disabled state due to a motion done timeout during clamped
<b>60</b>	NOTINIT state due to a group interlock error on not reference state
<b>61</b>	NOTINIT state due to a group interlock error during homing
<b>63</b>	NOTINIT state due to a motor initialization error
<b>64</b>	Referencing state
<b>65</b>	Clamping initialization
<b>66</b>	NOTINIT state due to a perpendicularity error homing
<b>67</b>	NOTINIT state due to a master/slave error during homing
<b>68</b>	Auto-tuning state
<b>69</b>	Scaling calibration state
<b>70</b>	Ready state from auto-tuning
<b>71</b>	NOTINIT state from scaling calibration
<b>72</b>	NOTINIT state due to a scaling calibration error
<b>73</b>	Excitation signal generation state
<b>74</b>	Disable state due to a following error on excitation signal generation state
<b>75</b>	Disable state due to a master/slave error on excitation signal generation state
<b>76</b>	Disable state due to an emergency stop on excitation signal generation state
<b>77</b>	Ready state from excitation signal generation
<b>78</b>	Focus state
<b>79</b>	Ready state from focus
<b>80</b>	Disable state due to a following error on focus state
<b>81</b>	Disable state due to a master/slave error on focus state
<b>82</b>	Disable state due to an emergency stop on focus state
<b>83</b>	NOTINIT state due to a group interlock error
<b>84</b>	Disable state due to a group interlock error during moving
<b>85</b>	Disable state due to a group interlock error during jogging
<b>86</b>	Disable state due to a group interlock error on slave state
<b>87</b>	Disable state due to a group interlock error during trajectory
<b>88</b>	Disable state due to a group interlock error during analog tracking
<b>89</b>	Disable state due to a group interlock error during spinning
<b>90</b>	Disable state due to a group interlock error on ready state
<b>91</b>	Disable state due to a group interlock error on auto-tuning state
<b>92</b>	Disable state due to a group interlock error on excitation signal generation state
<b>93</b>	Disable state due to a group interlock error on focus state
<b>94</b>	Disabled state due to a motion done timeout during jogging
<b>95</b>	Disabled state due to a motion done timeout during spinning

<b>96</b>	Disabled state due to a motion done timeout during slave mode
<b>97</b>	Disabled state due to a ZYGO error during motion
<b>98</b>	Disabled state due to a master/slave error during trajectory
<b>99</b>	Disable state due to a ZYGO error on jogging state
<b>100</b>	Disabled state due to a ZYGO error during analog tracking
<b>101</b>	Disable state due to a ZYGO error on auto-tuning state
<b>102</b>	Disable state due to a ZYGO error on excitation signal generation state
<b>103</b>	Disabled state due to a ZYGO error on ready state
<b>104</b>	Driver initialization
<b>105</b>	Jitter initialization
<b>106</b>	Not initialized state due to an error with GroupKill or KillAll command

## 7.9 Error List

code	Error description
<b>2</b>	Error to ignore
<b>1</b>	TCL interpreter error: wrong syntax
<b>0</b>	Successful command
<b>-1</b>	Busy socket: previous command not yet finished
<b>-2</b>	TCP timeout
<b>-3</b>	String command too long
<b>-4</b>	Unknown command
<b>-5</b>	Not allowed due to a positioner error
<b>-7</b>	Wrong format in the command string
<b>-8</b>	Wrong object type for this command
<b>-9</b>	Wrong number of parameters in the command
<b>-10</b>	Wrong parameter type in the command string
<b>-11</b>	Wrong parameters type in the command string: word or word * expected
<b>-12</b>	Wrong parameter type in the command string: bool or bool * expected
<b>-13</b>	Wrong parameter type in the command string: char * expected
<b>-14</b>	Wrong parameter type in the command string: double or double * expected
<b>-15</b>	Wrong parameter type in the command string: int or int * expected
<b>-16</b>	Wrong parameter type in the command string: unsigned int or unsigned int * expected
<b>-17</b>	Parameter out of range
<b>-18</b>	Positioner Name doesn't exist
<b>-19</b>	GroupName doesn't exist or unknown command
<b>-20</b>	Fatal Error during initialization, read the error.log file for more details
<b>-21</b>	Controller in initialization
<b>-22</b>	Not allowed action
<b>-23</b>	Position compare not set
<b>-24</b>	Not available in this configuration
<b>-25</b>	Following Error
<b>-26</b>	Emergency signal
<b>-27</b>	Move Aborted
<b>-28</b>	Home search timeout
<b>-29</b>	Mnemonic gathering type doesn't exist
<b>-30</b>	Gathering not started
<b>-31</b>	Home position is out of user travel limits
<b>-32</b>	Gathering not configurated
<b>-33</b>	Motion done timeout
<b>-35</b>	Not allowed: home preset outside travel limits
<b>-36</b>	Unknown TCL file
<b>-37</b>	TCL interpreter doesn't run
<b>-38</b>	TCL script can't be killed
<b>-39</b>	Mnemonic action doesn't exist
<b>-40</b>	Mnemonic event doesn't exist
<b>-41</b>	Slave-Master mode not configurated
<b>-42</b>	Jog value out of range

<b>-43</b>	Gathering running
<b>-44</b>	Slave error disabling master
<b>-45</b>	End of run activated
<b>-46</b>	Not allowed action due to backlash
<b>-47</b>	Wrong TCL task name: each TCL task name must be different
<b>-48</b>	BaseVelocity must be null
<b>-49</b>	Inconsistent mechanical zero during home search
<b>-50</b>	Motor initialization error: check InitializationAcceleration
<b>-51</b>	Spin value out of range
<b>-52</b>	Group interlock
<b>-53</b>	Not allowed action due to a group interlock
<b>-60</b>	Error during file writing or file doesn't exist
<b>-61</b>	Error during file reading or file doesn't exist
<b>-62</b>	Wrong trajectory element type
<b>-63</b>	Wrong XY trajectory element arc radius
<b>-64</b>	Wrong XY trajectory element sweep angle
<b>-65</b>	Trajectory line element discontinuity error or new element is too small
<b>-66</b>	Trajectory doesn't content any element or not loaded
<b>-68</b>	Velocity on trajectory is too high
<b>-69</b>	Acceleration on trajectory is too high
<b>-70</b>	Final velocity on trajectory is not zero
<b>-71</b>	Error write or read from message queue
<b>-72</b>	Error during trajectory initialization
<b>-73</b>	End of file
<b>-74</b>	Error file parameter key not found
<b>-75</b>	Time delta of trajectory element is negative or null
<b>-80</b>	Event not configured
<b>-81</b>	Action not configured
<b>-82</b>	Event buffer is full
<b>-83</b>	Event ID not defined
<b>-85</b>	Secondary positioner index is too far from first positioner
<b>-90</b>	Focus socket not reseved or closed
<b>-91</b>	Focus event scheduler is busy
<b>-95</b>	Error of executing an optional module
<b>-96</b>	Error of stopping an optional module
<b>-98</b>	Error of unloading an optional module
<b>-99</b>	Fatal external module load: see error.log
<b>-100</b>	Internal error (memory allocation error, ...)
<b>-101</b>	Relay Feedback Test failed: No oscillation
<b>-102</b>	Relay Feedback Test failed: Signal too noisy
<b>-103</b>	Relay Feedback Test failed: Signal data not enough for analyse
<b>-104</b>	Error of tuning process initialization
<b>-105</b>	Error of scaling calibration initialization
<b>-106</b>	Wrong user name or password
<b>-107</b>	This function requires to be logged in with Administrator rights
<b>-108</b>	The TCP/IP connection was closed by an administrator
<b>-109</b>	Group need to be homed at least once to use this function (distance mesured during home search)

<b>-110</b>	Execution not allowed for Gantry configuration
<b>-111</b>	Gathering buffer is full
<b>-112</b>	Error of excitation signal generation initialization
<b>-113</b>	Both ends of run activated
<b>-114</b>	Clamping timeout
<b>-115</b>	Function is not supported by current hardware
<b>-116</b>	Error during external driver initialization, read error.log file for more details
<b>-117</b>	Function is only allowed in DISABLED group state
<b>-118</b>	Not allowed action driver not initialized
<b>-119</b>	Position is outside of travel limits on secondary positioner
<b>-120</b>	Warning following error during move with position compare enabled
<b>-121</b>	Function is not allowed due to configuration disabled
<b>-122</b>	Data incorrect (wrong value, wrong format, wrong order or nonexistent)
<b>-123</b>	Action not allowed, an Administrator is already logged in
<b>-124</b>	Error during move of secondary positioner: check positioners errors for details
<b>-125</b>	Check tcl task name is not empty
<b>-126</b>	Wrong parameter type in the command string: short or short * expected
<b>-127</b>	Wrong parameter type in the command string: long or long * expected
<b>-128</b>	Wrong parameter type in the command string: unsigned short or unsigned short * expected
<b>-129</b>	Wrong parameter type in the command string: unsigned long or unsigned long * expected
<b>-130</b>	Wrong parameter type in the command string: float or float * expected
<b>-131</b>	Wrong parameter type in the command string: long long int or long long int * expected
<b>-132</b>	Wrong parameter type in the command string: unsigned long long or unsigned long long * expected
<b>-133</b>	Error when creating actions tasks
<b>-134</b>	Changing the loop status is allowed in DISABLE state only
<b>-135</b>	Function is not allowed because group is not initialized or not referenced
<b>-136</b>	Wrong parameter type in the command string: charhex32 * expected
<b>-137</b>	Event&Action: action threads number exceeds limit (must be $\leq 20$ )
<b>-138</b>	Event&Action: action thread is running
<b>-139</b>	Event&Action: Always event is not compatible with associated action
<b>-200</b>	Invalid socket
<b>-201</b>	The group is already in this mode
<b>-202</b>	Not allowed action due to an external motion interlock
<b>-204</b>	Function is not allowed because the feed forward is enabled
<b>-205</b>	Not enable in your configuration
<b>-206</b>	Dual encoder position error
<b>-207</b>	Gantry mode error: check Encoder Matrix and Decoupling Motor Matrix
<b>-208</b>	Not allowed action because piston is engaged
<b>-209</b>	INT board command failed: invalid card number or initialization not done
<b>-210</b>	Not allowed action due to ( $XStart \leq Xangle \leq XEnd$ ) is not true

<b>-211</b>	Not expected position after motion
<b>-212</b>	MagneticTrackPositionAtHome value is out of tolerance and can not be applied.
<b>-1000</b>	Zygo command execution failed
<b>-1001</b>	The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
<b>-1002</b>	Connection to Zygo TCP server failed
<b>-1003</b>	The XPS controller is already connected to Zygo TCP server
<b>-1004</b>	Zygo signal is not present
<b>-1005</b>	Zygo PEG configuration failed
<b>-1006</b>	Zygo error detected

## 7.10 Controller Status List

Controller status code	code	Controller status description
CONTROLLER_STATUS_OK	0x00000000	Controller status OK
CONTROLLER_STATUS_INITIALIZATION_FAILED	0x00000001	Controller initialization failed
CONTROLLER_STATUS_NB_OPENED_SOCKETS_REACHED_MAXIMUM_ALLOWED	0x00000002	Number of currently opened sockets reached maximum allowed number
CONTROLLER_STATUS_CPU_OVERLOAD	0x00000004	Controller CPU is overloaded
CONTROLLER_STATUS_CORRECTOR_OVER_CALCULATED	0x00000008	Current measured corrector calculation time exceeds the corrector period
CONTROLLER_STATUS_PROFILER_OVER_CALCULATED	0x00000010	Profile generator calculating time exceeds ProfileGeneratorISRRatio * IRSCorrectorPeriod
CONTROLLER_STATUS_CORRECTOR_INTERRUPT_LOST	0x00000020	Controller has lost a corrector interrupt
CONTROLLER_STATUS_INTERFERO_SIGNAL_NOT_PRESENT	0x00000040	Zygo interferometer signal is not present
CONTROLLER_STATUS_INTERFERO_INITIALIZATION_FAILED	0x00000080	Zygo interferometer Ethernet initialisation failed
CONTROLLER_STATUS_INTERFERO_ERROR_STATUS	0x00000100	Zygo interferometer error detected. Please check ZYGO Error Status
CONTROLLER_STATUS_MOTION_VELOCITY_LIMITED	0x00000200	Motion velocity is limited
CONTROLLER_STATUS_LIFT_PIN_UP	0x00000400	Lift pin is UP

### NOTE

Within about 5 minutes after the controller startup, due to the hardware thermal stabilization, the

**CONTROLLER\_STATUS\_CORRECTOR\_OVER\_CALCULATED,**

**CONTROLLER\_STATUS\_CORRECTOR\_INTERRUPT\_LOST ,**

**CONTROLLER\_STATUS\_PROFILER\_OVER\_CALCULATED,**

**CONTROLLER\_STATUS\_CPU\_OVERLOAD or**

**CONTROLLER\_STATUS\_NB\_OPENED\_SOCKETS\_REACHED\_MAXIMUM\_ALLOWED** status flags may be raised.

These flags are automatically reset after a controller status reading using the **ControllerStatusGet()** command.

Another way to avoid these flags during the 5 first minutes after boot is to set the following parameter in system.ref to 300 (seconds):

**DelayBeforeStartup = 300 ; Controller boots completely after 300 seconds**

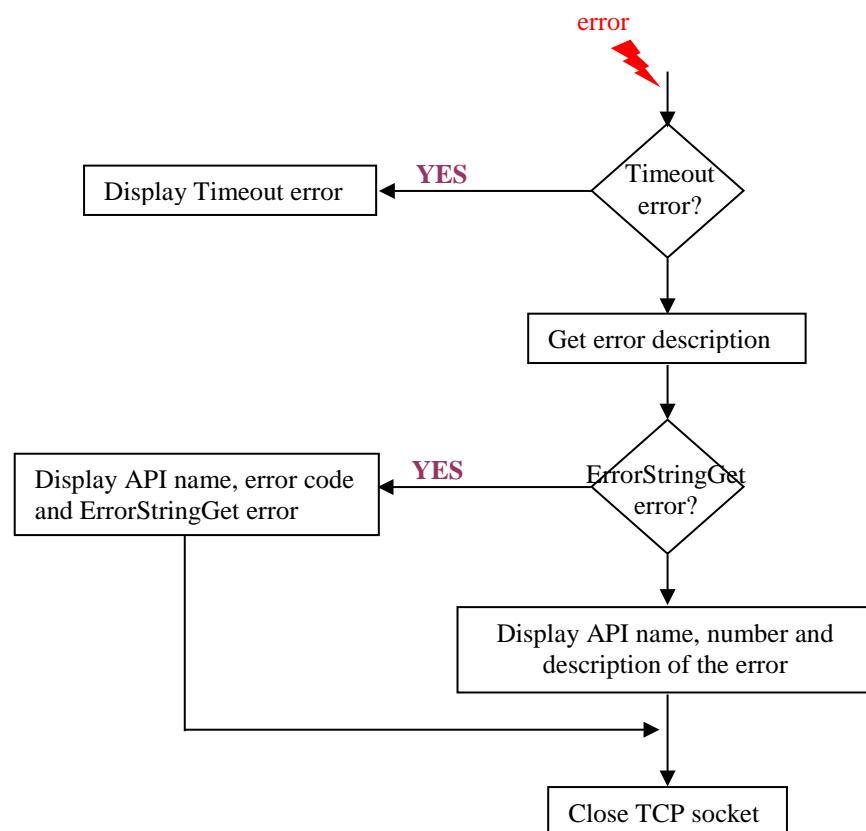
## 8. Process Examples

This section provides examples of programming sequences. The next diagrams show the order of use of the different Functions. To see programming code examples, please refer to the TCL Manual for TCL scripts.

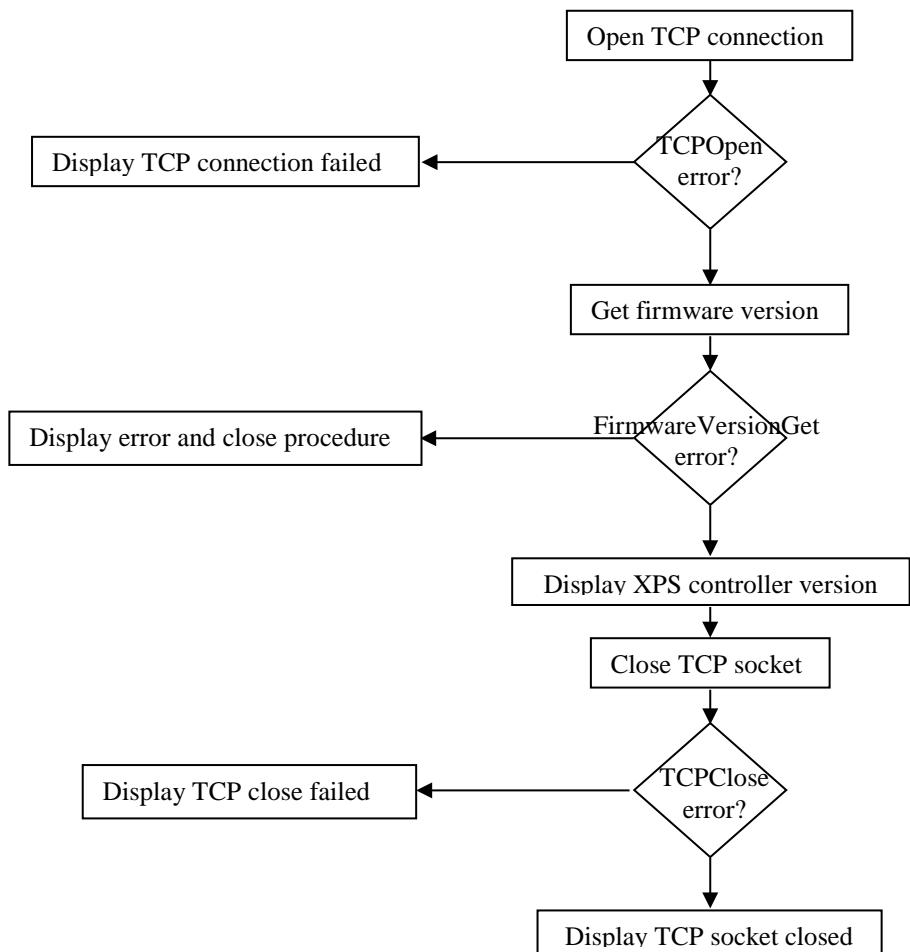
### 8.1 Management of Errors Example

When an error occurs, it is desirable to analyze and fix the error. The following error display and socket closing procedure is useful to detect and display the errors during the execution of a program. This sequence could be added to each program and called each time users need to test certain parts of a program.

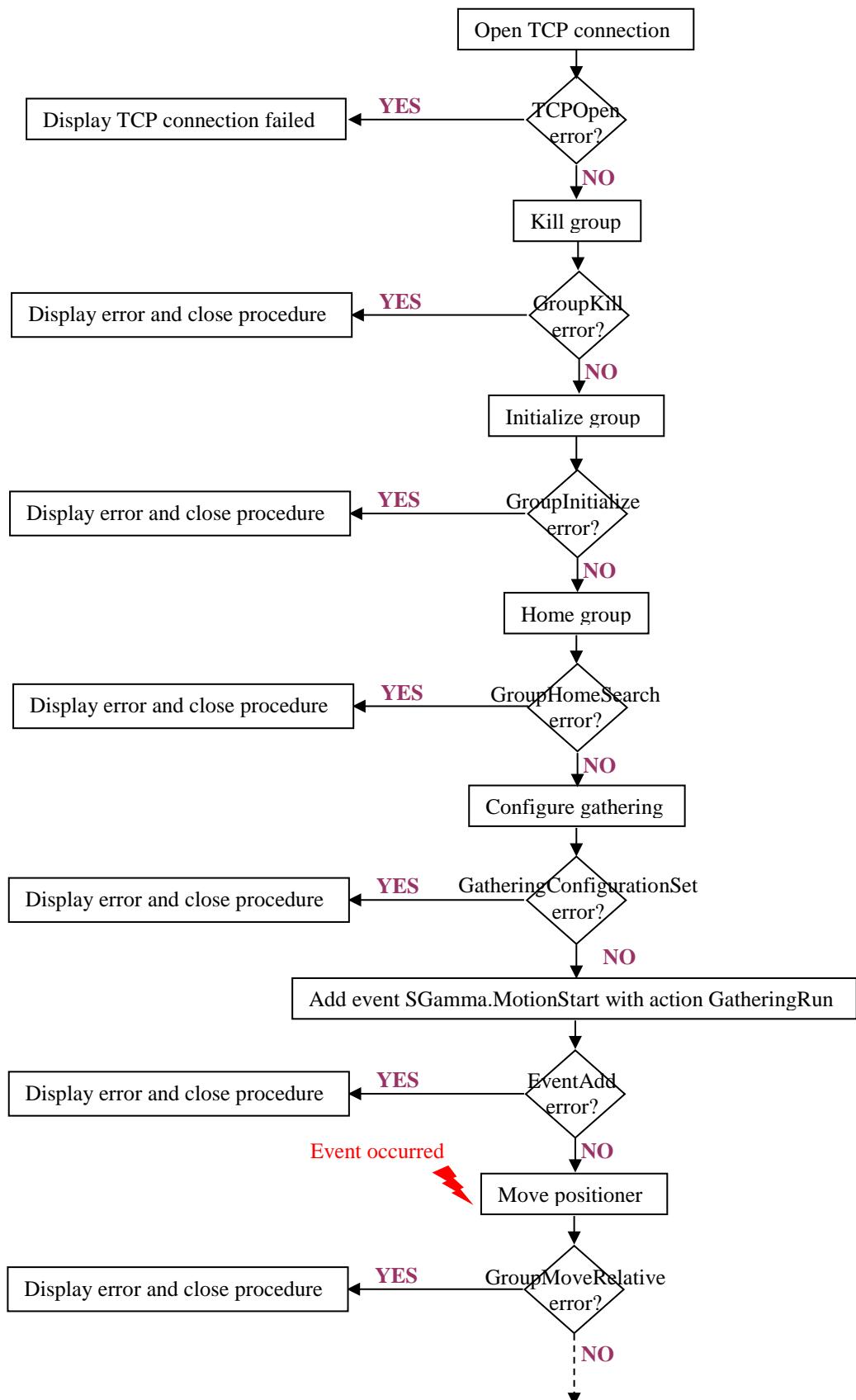
*Display error and close procedure:*

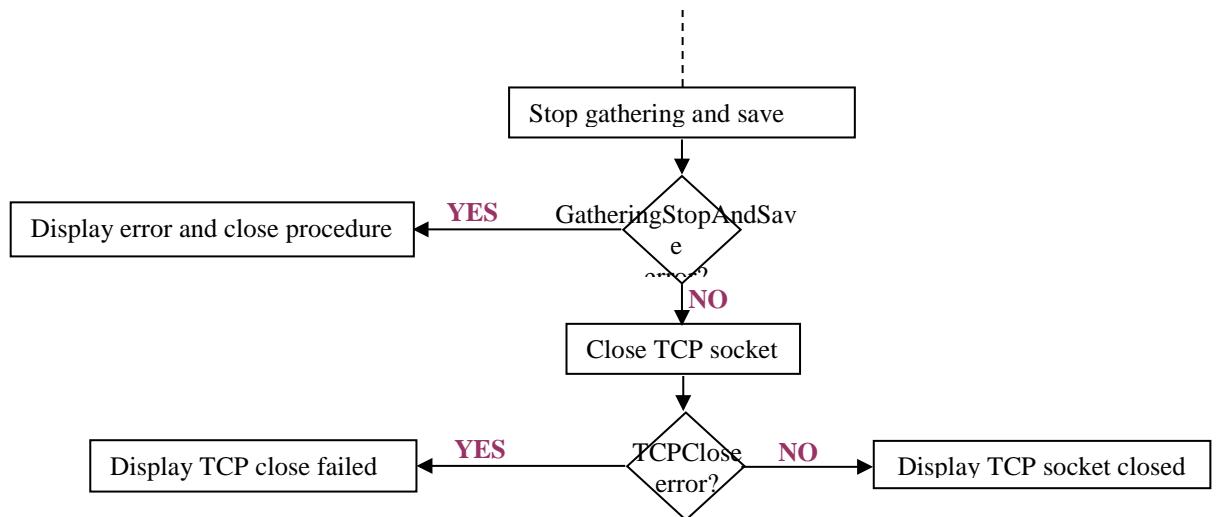


## 8.2 Firmware Version Example

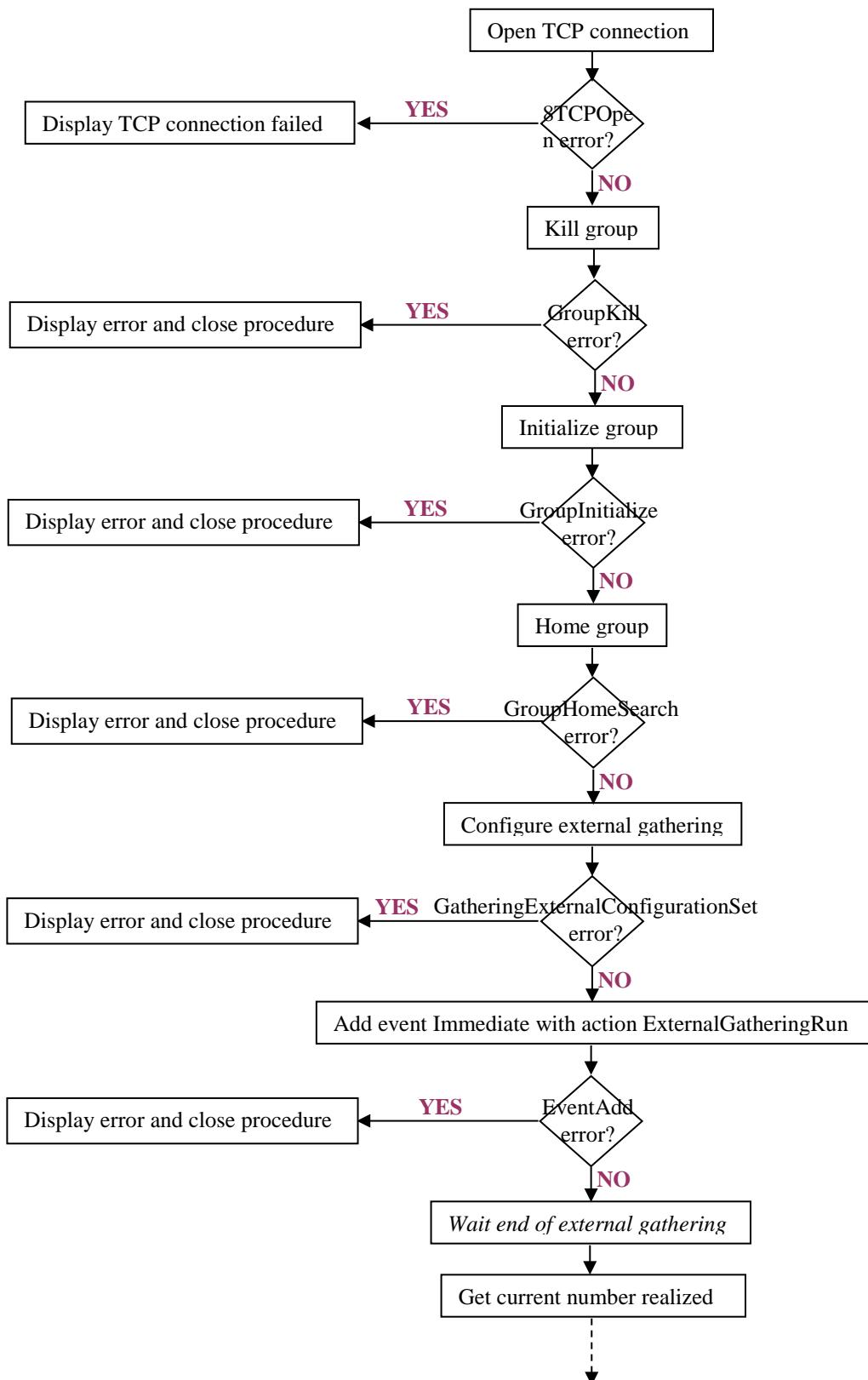


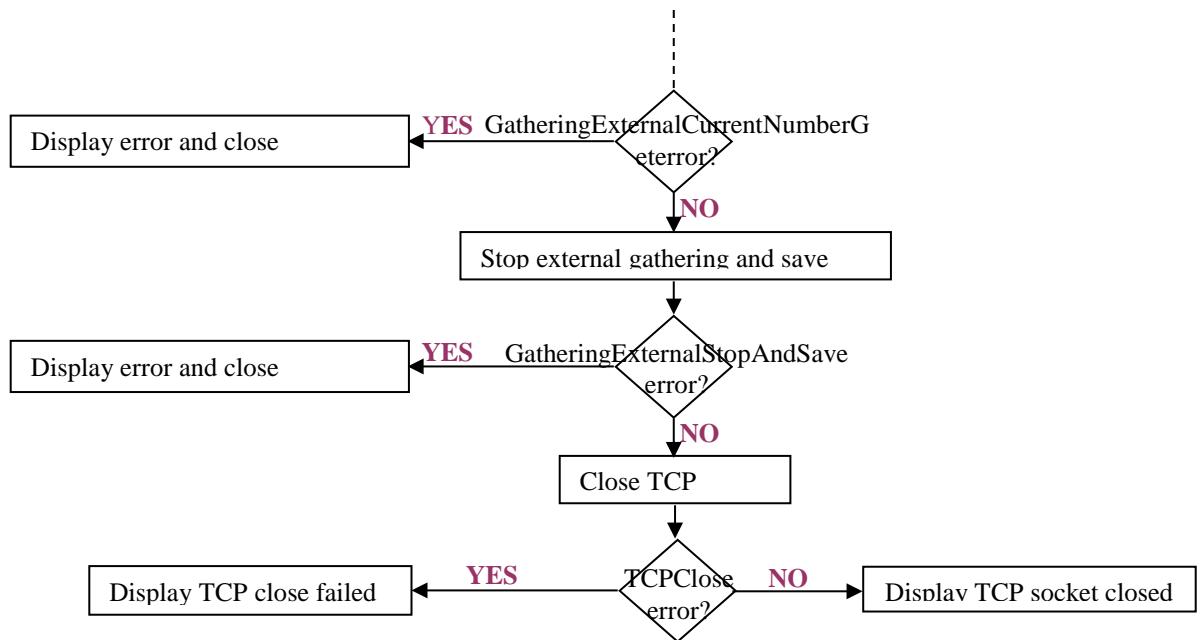
### 8.3 Gathering with Motion Example



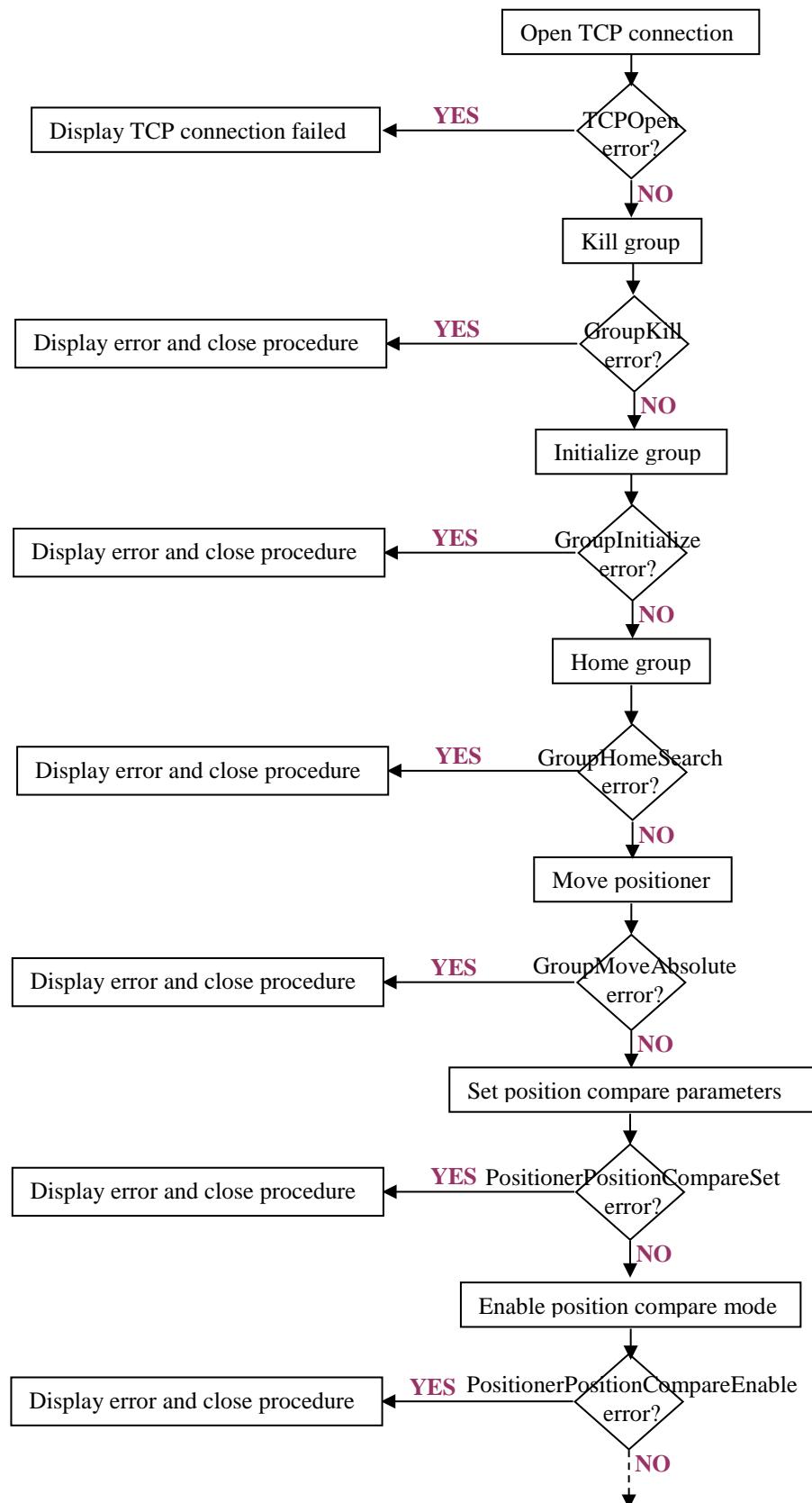


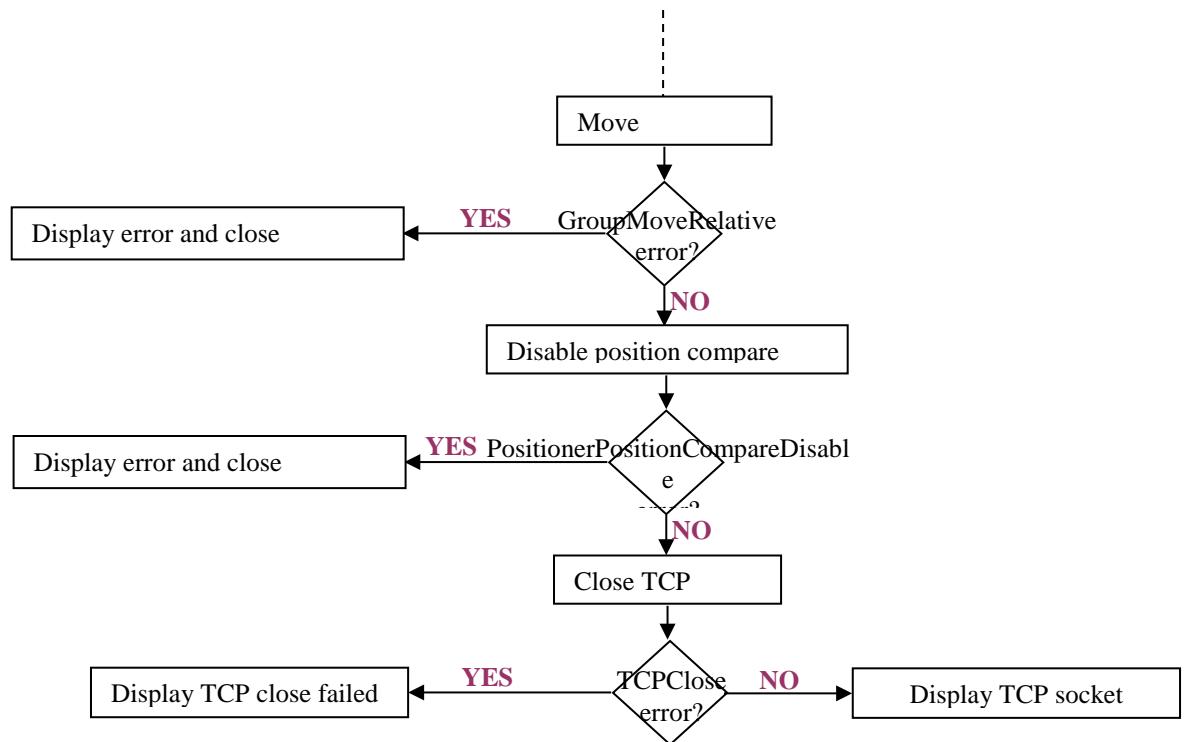
## 8.4 External Gathering Example



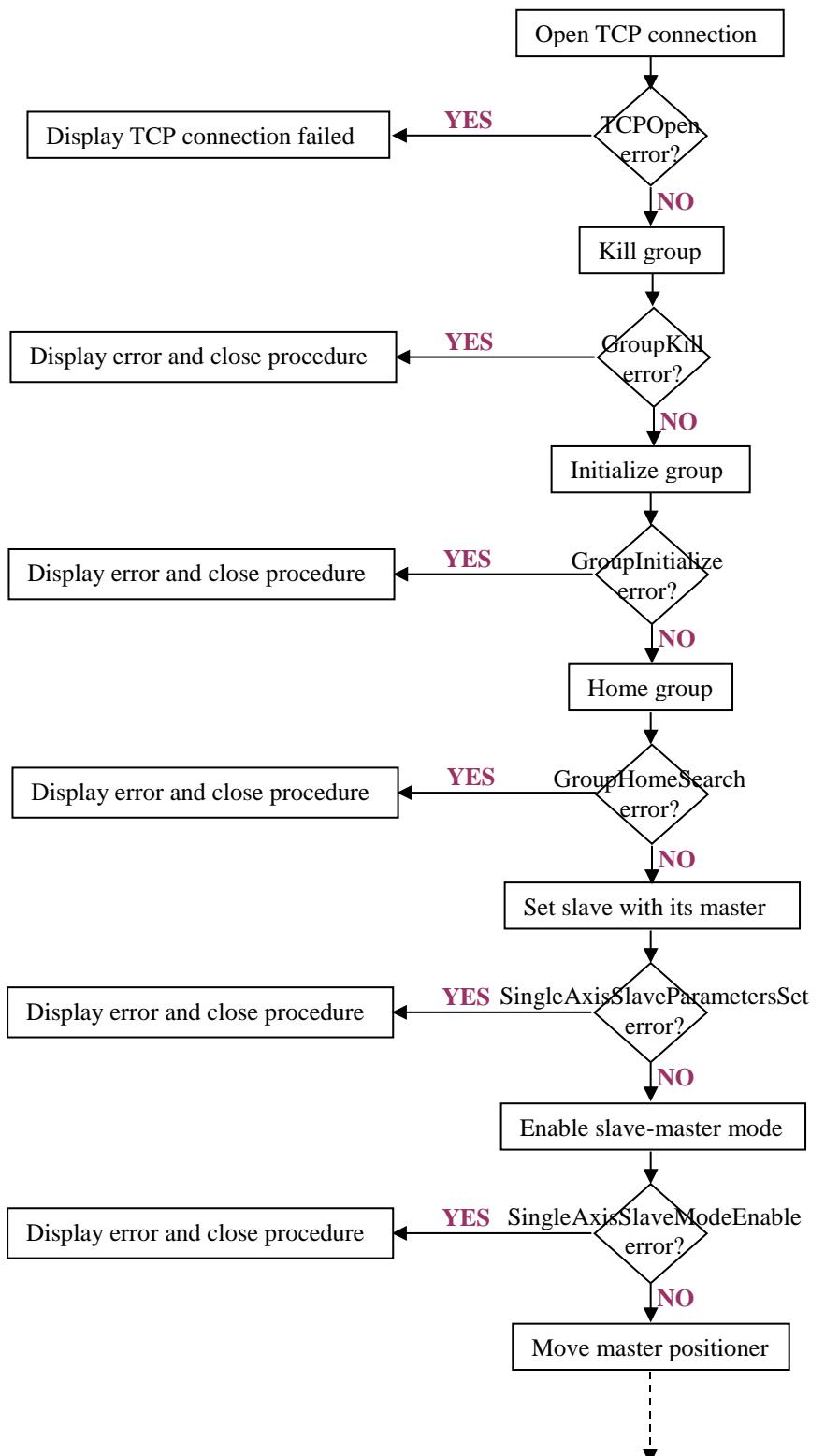


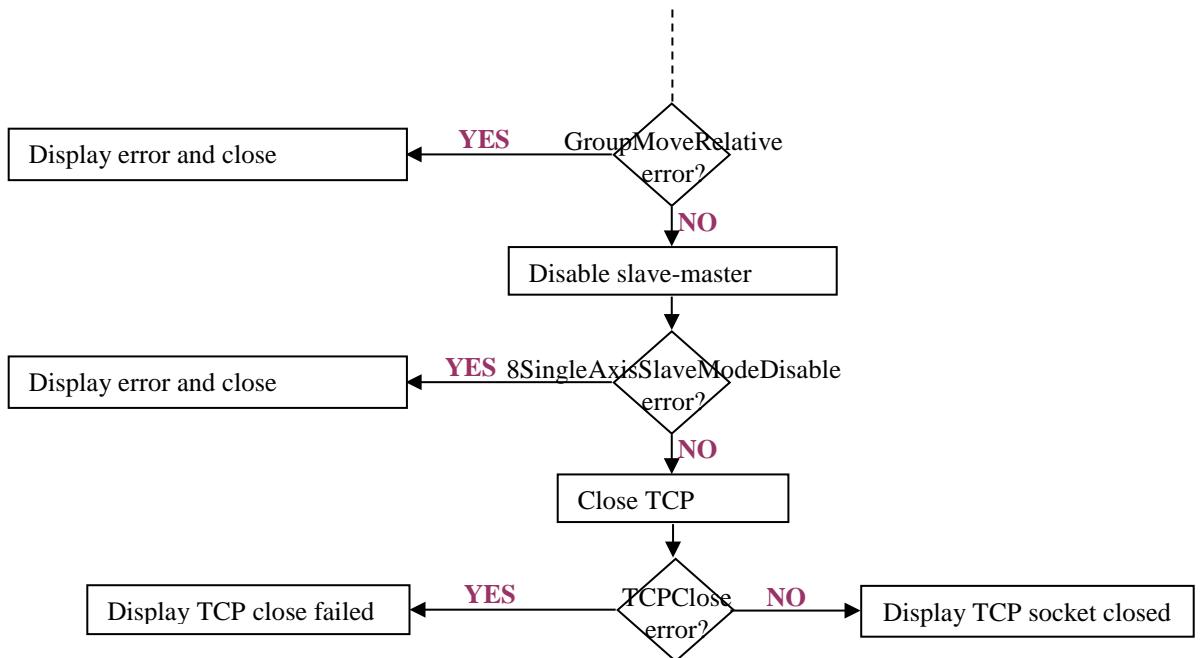
## 8.5 Position Output Compare Example



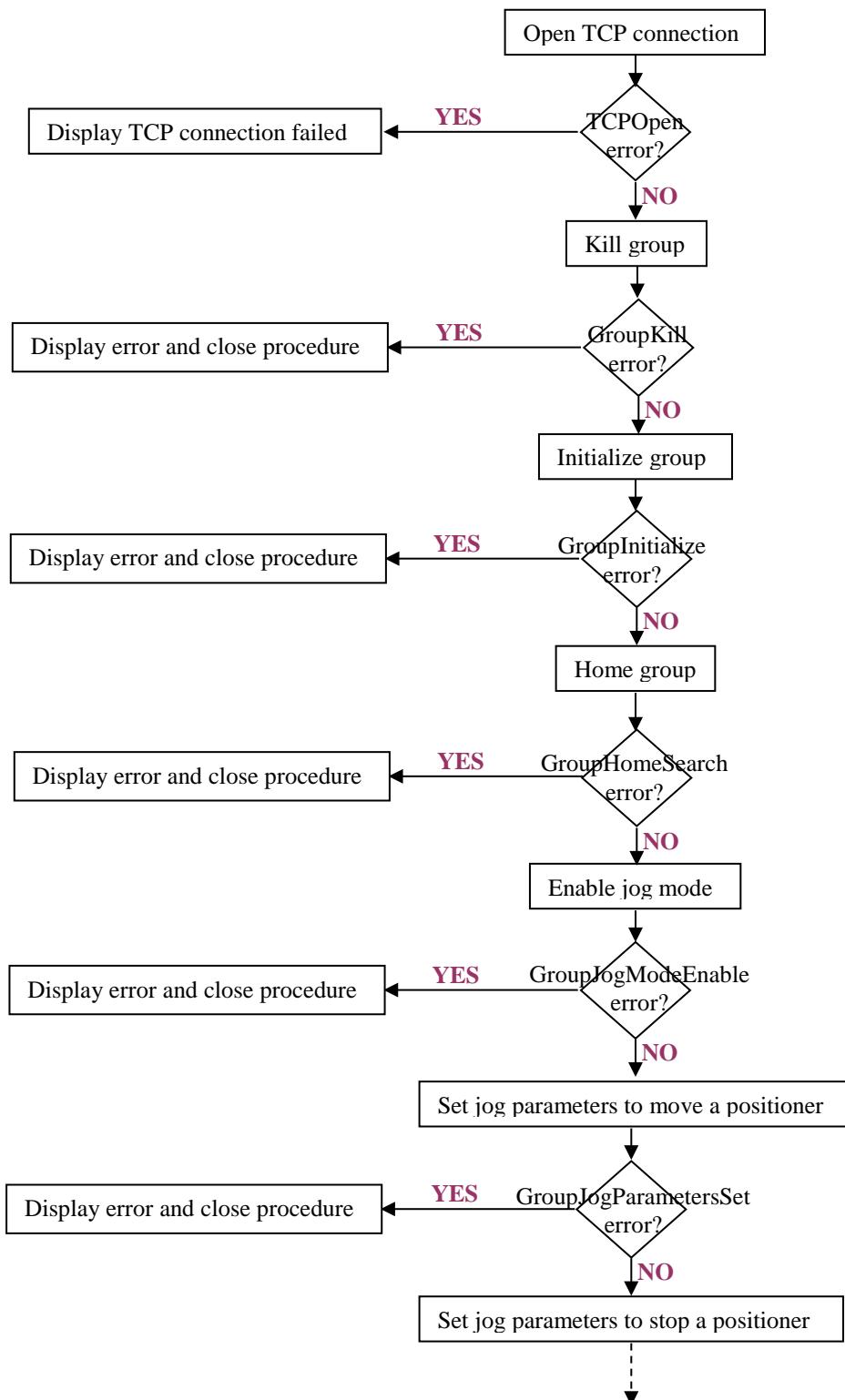


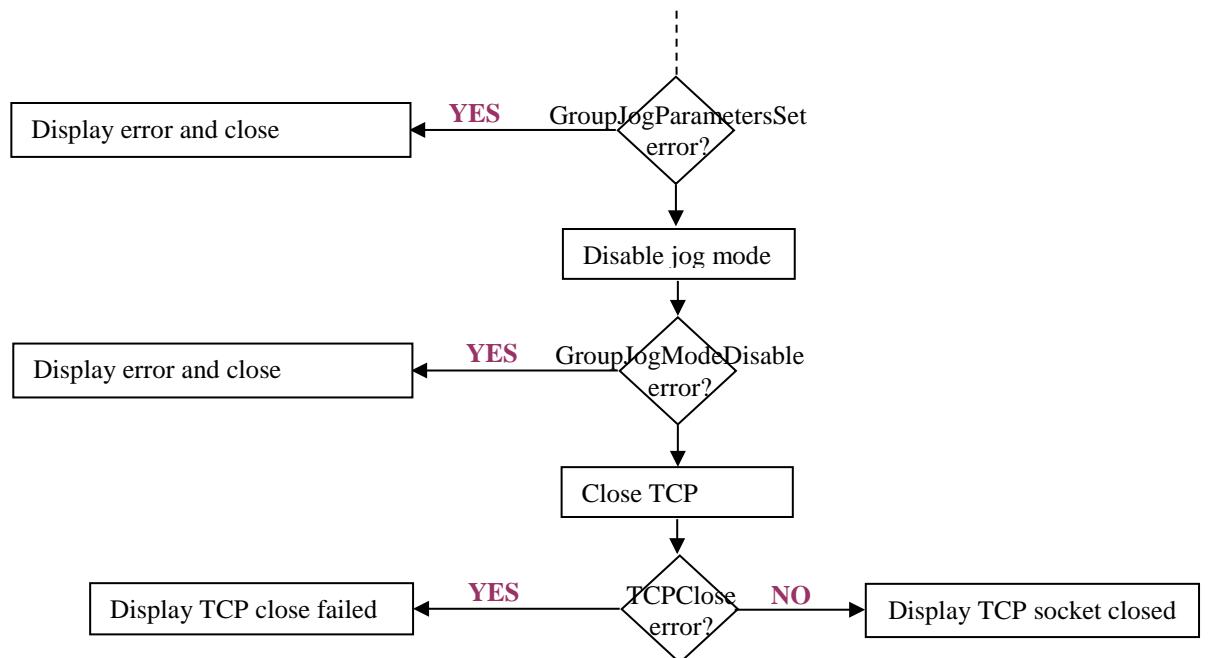
## 8.6 Slave-Master Mode Example



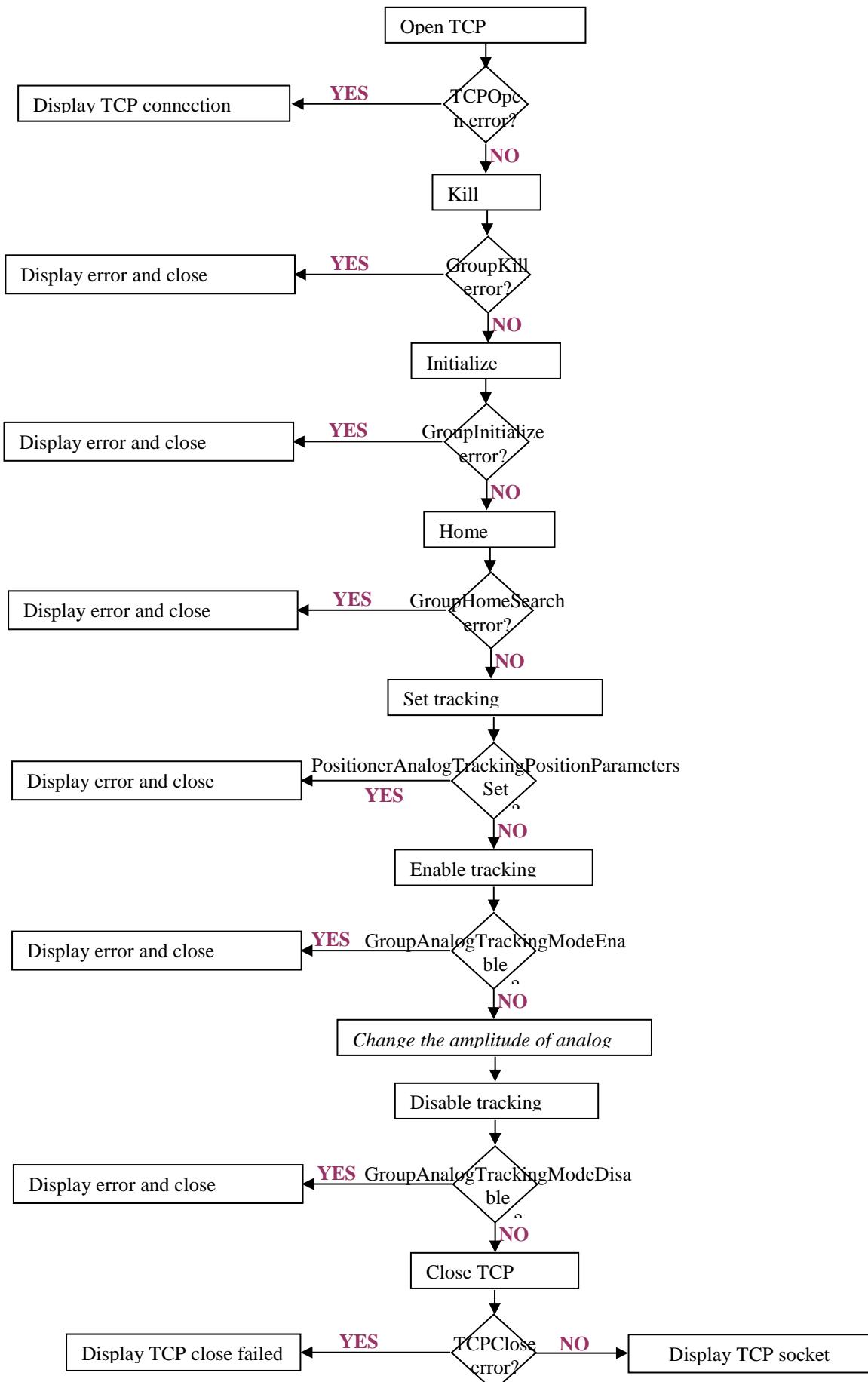


## 8.7 Jogging Example

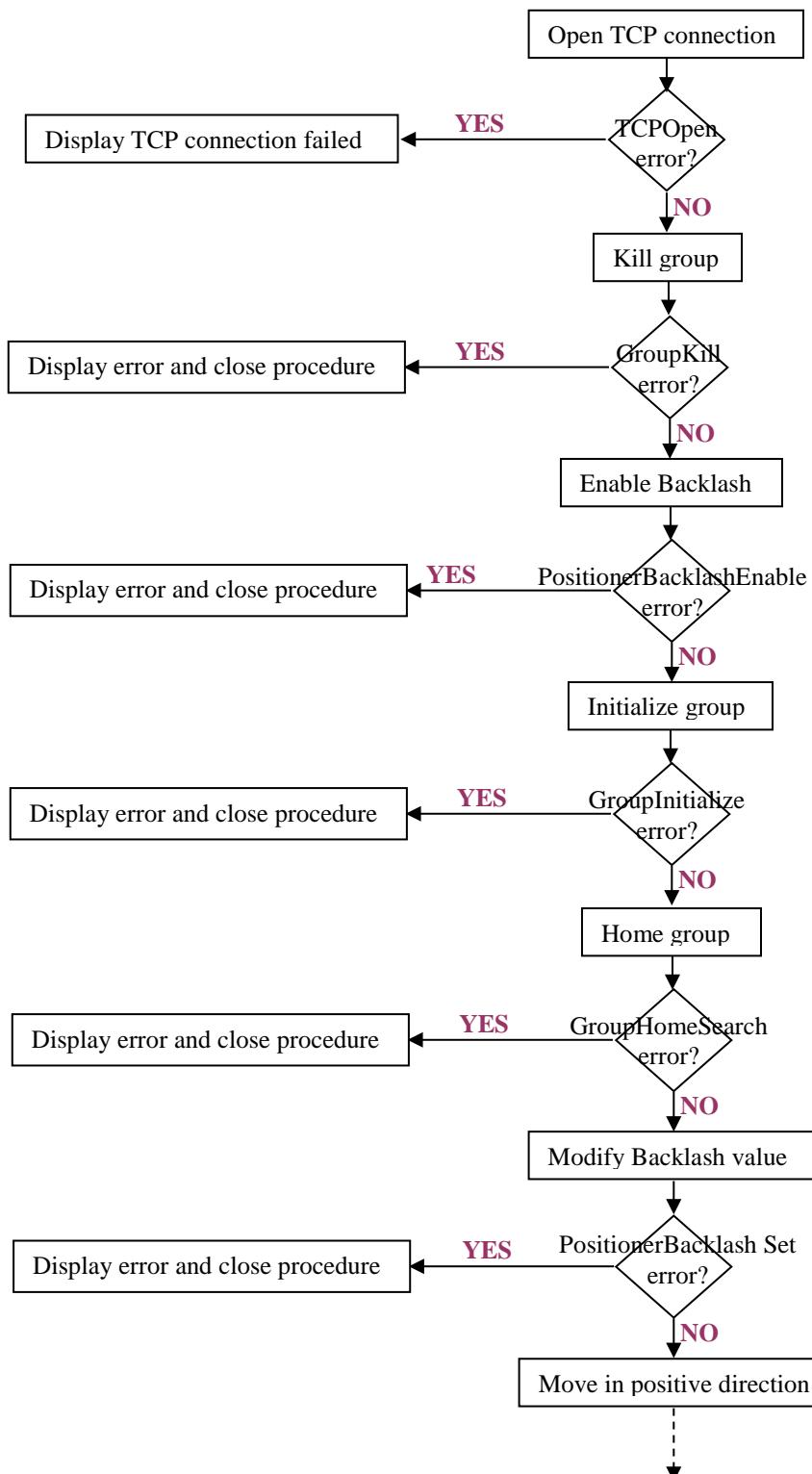


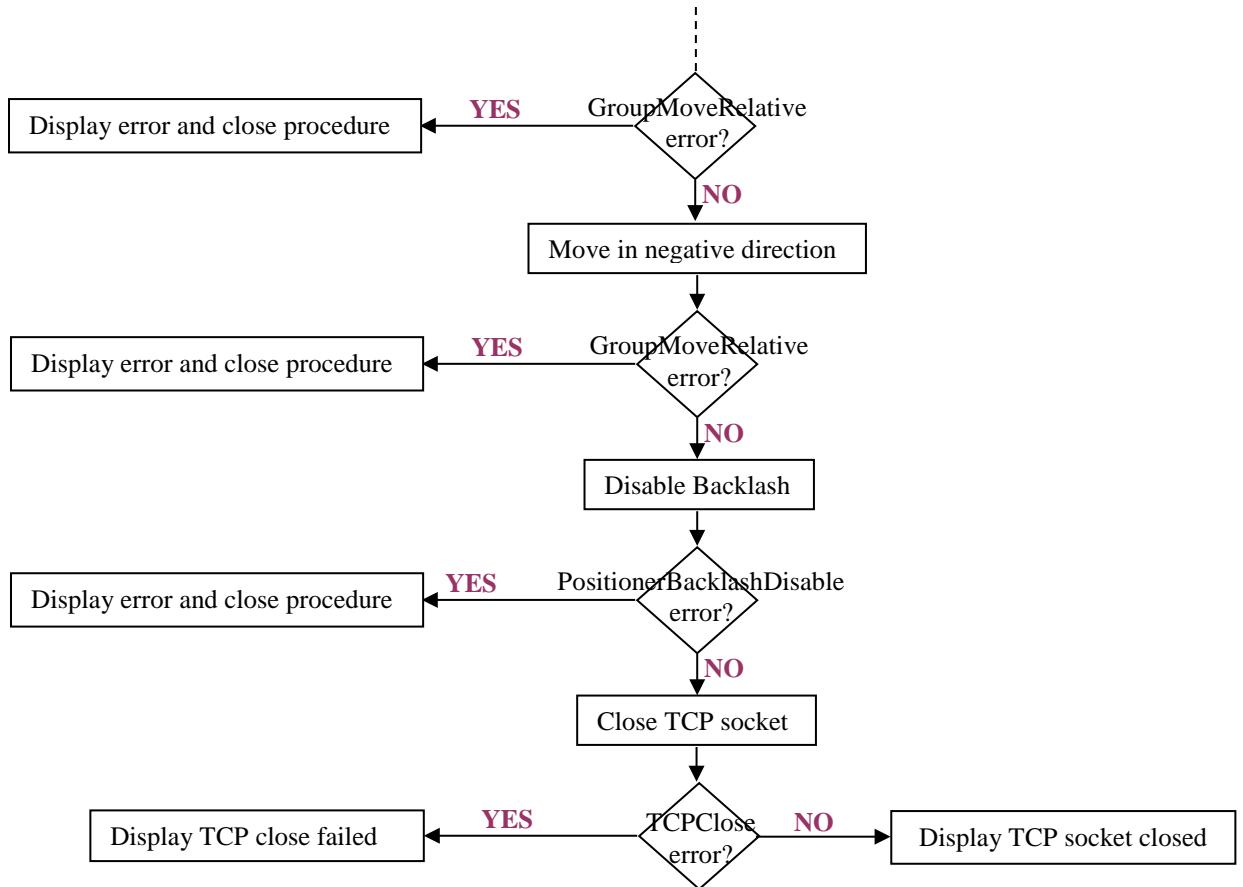


## 8.8 Tracking Example

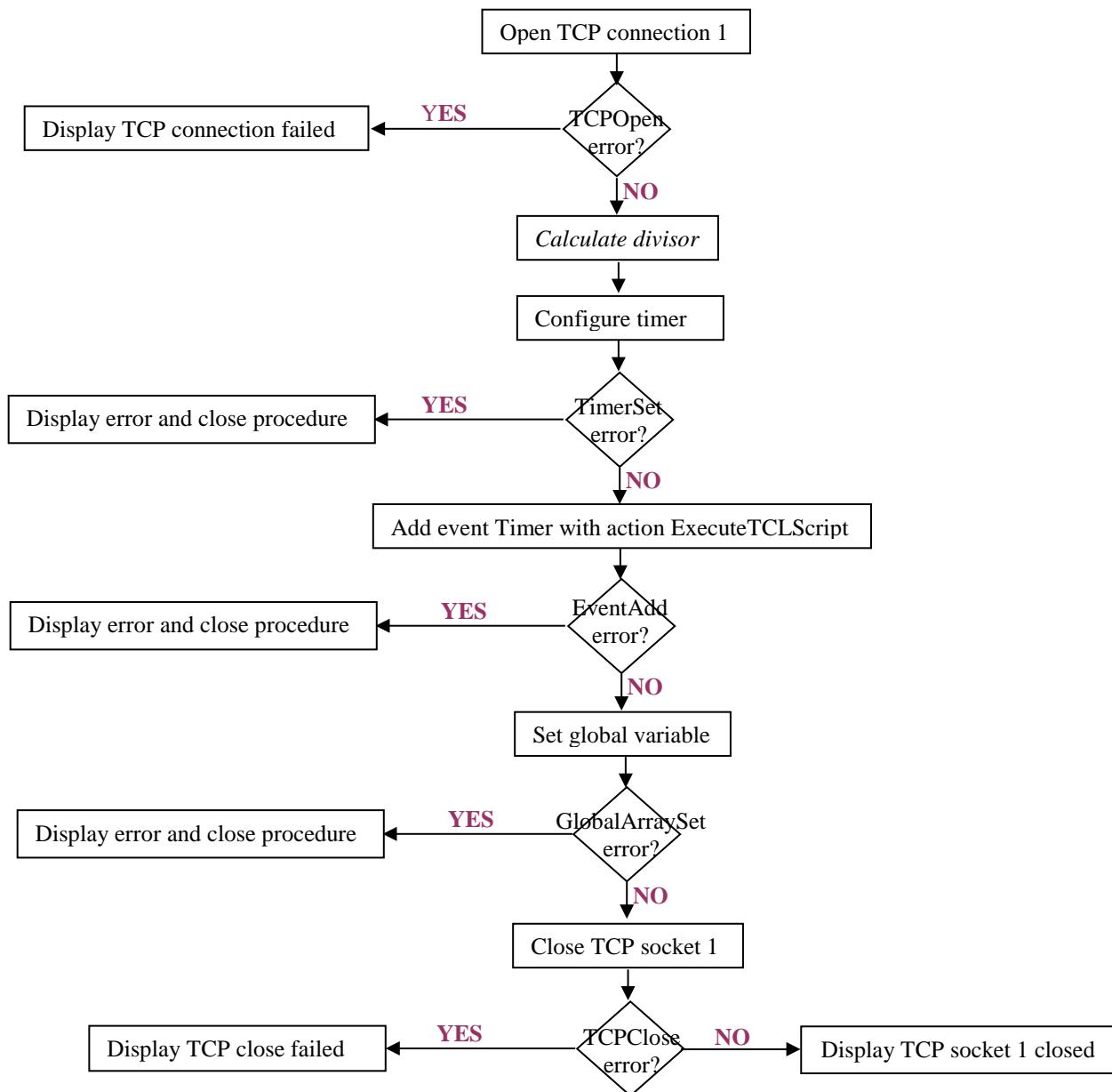


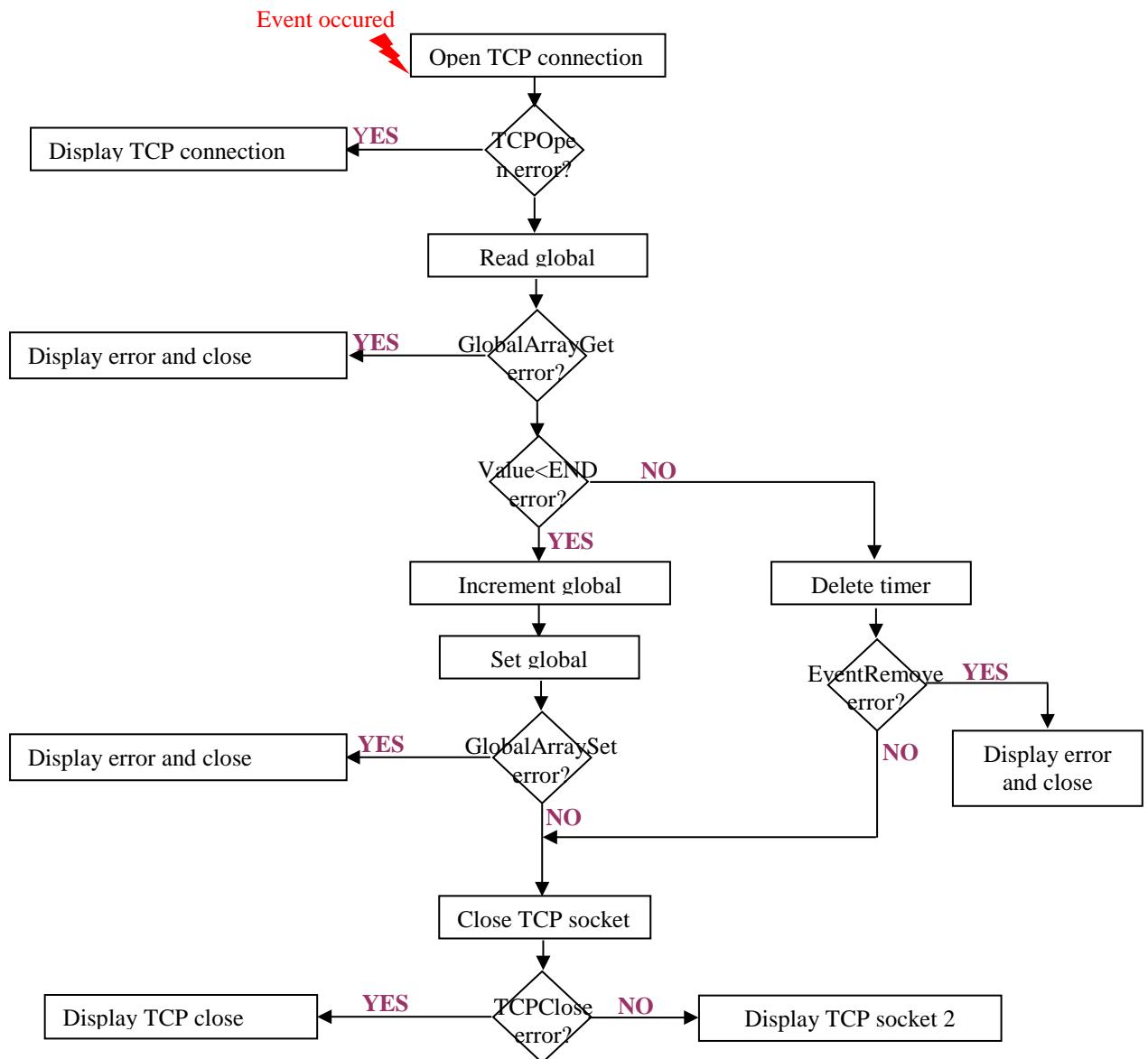
## 8.9 Backlash



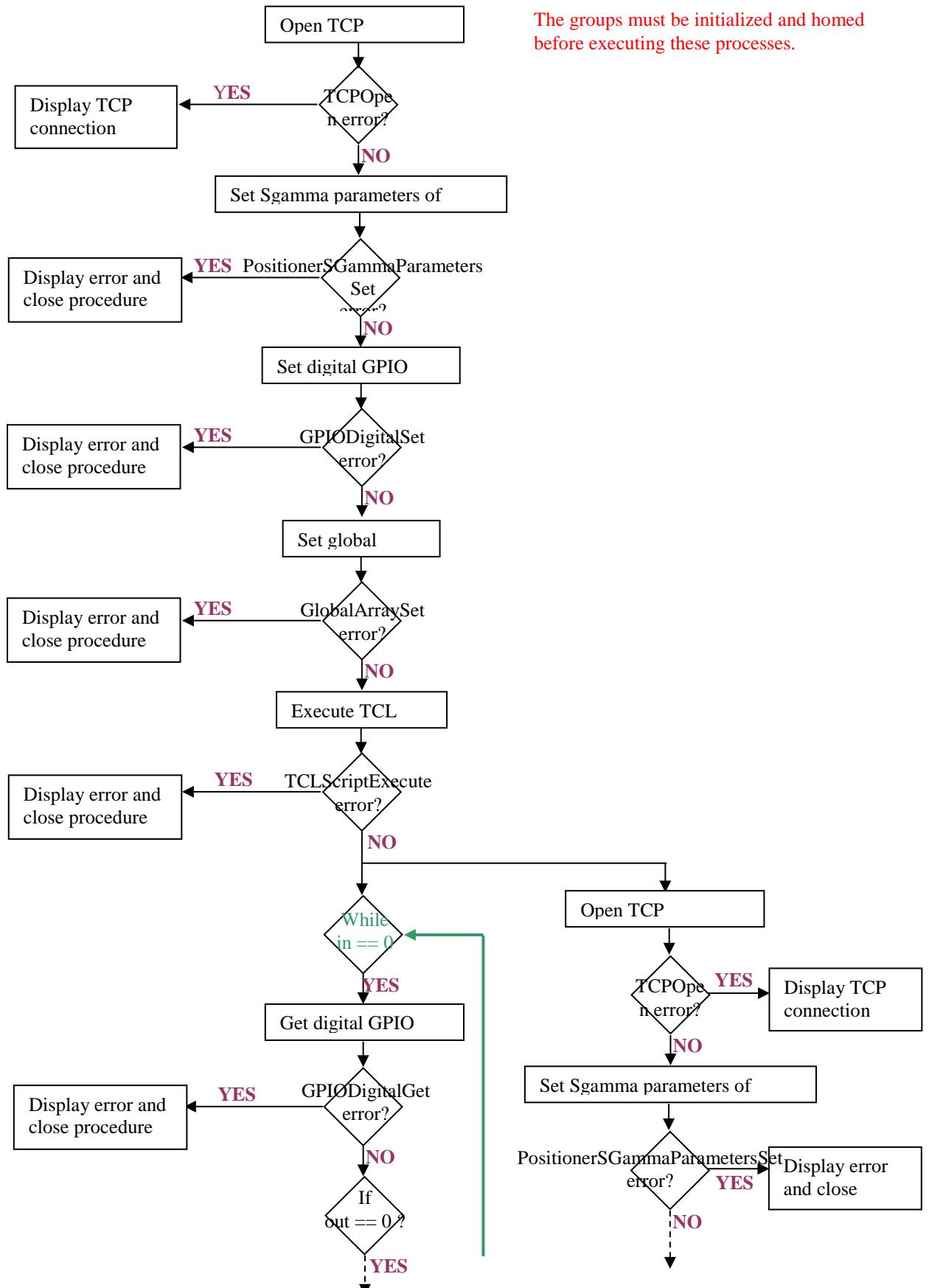


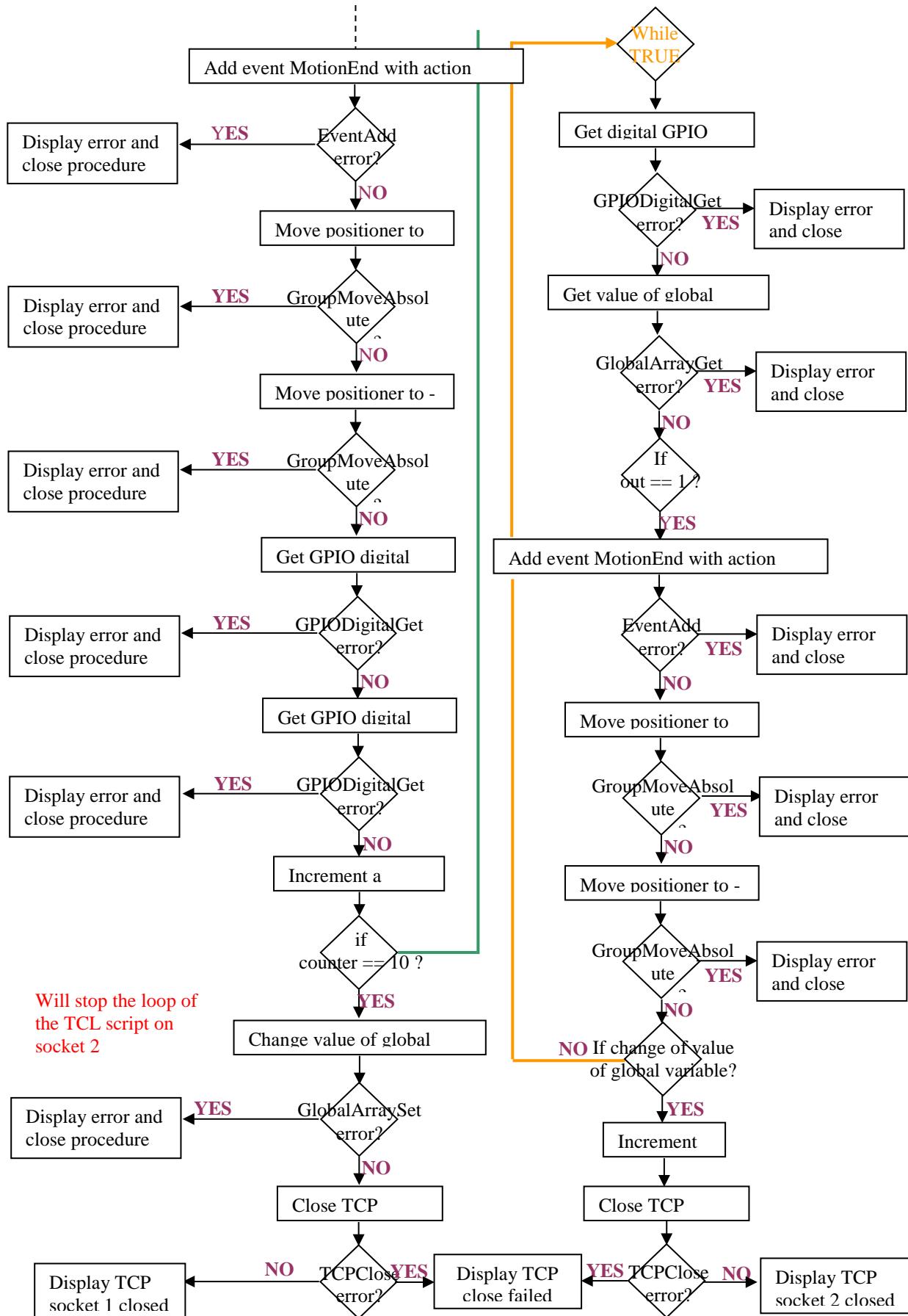
## 8.10 Timer Event and Global Variables





## 8.11 Running Several Motion Processes Simultaneously





## Service Form

## Your Local Representative

Tel.: \_\_\_\_\_

Fax: \_\_\_\_\_

Name: \_\_\_\_\_

Return authorization #: \_\_\_\_\_

*(Please obtain prior to return of item)*

Company: \_\_\_\_\_

Date: \_\_\_\_\_

Address: \_\_\_\_\_

Phone Number: \_\_\_\_\_

Country: \_\_\_\_\_

Fax Number: \_\_\_\_\_

P.O. Number: \_\_\_\_\_

Item(s) Being Returned: \_\_\_\_\_

Serial #: \_\_\_\_\_

Description: \_\_\_\_\_

Reasons of return of goods (please list any specific problems): \_\_\_\_\_



# Newport®

Experience | Solutions

Visit Newport Online at:  
[www.newport.com](http://www.newport.com)

#### **North America & Asia**

Newport Corporation  
1791 Deere Ave.  
Irvine, CA 92606, USA

#### **Sales**

Tel.: (800) 222-6440  
e-mail: sales@newport.com

#### **Technical Support**

Tel.: (800) 222-6440  
e-mail: tech@newport.com

**Service, RMAs & Returns**

Tel.: (800) 222-6440  
e-mail: service@newport.com

#### **Europe**

MICRO-CONTROLE Spectra-Physics S.A.S  
9, rue du Bois Sauvage  
91055 Évry CEDEX  
France

#### **Sales**

Tel.: +33 (0)1.60.91.68.68  
e-mail: france@newport.com

#### **Technical Support**

e-mail: tech\_europe@newport.com

#### **Service & Returns**

Tel.: +33 (0)2.38.40.51.55