XPS Unified

Universal High-Performance Motion Controller/Driver

Newport® Programmer’s Manual
V1.0.x
# Table of Contents

1.0 Note .................................................................................................................................................. 1

2.0 TCP/IP Communication .................................................................................................................. 2

3.0 XPS .NET Software Drivers ........................................................................................................... 3
   3.1 How to Install .NET Drivers for XPS Controller ................................................................. 3
      3.1.1 Requirements ............................................................................................................... 3
      3.1.1 Installing the 32 bit (x86) Windows Platform .......................................................... 4
      3.1.2 Installing the 64 bit (x64) Windows Platform .......................................................... 4
   3.2 Variables Equivalent for Programming Languages ............................................................... 5
   3.3 How to Use XPS .NET Assembly from Visual Studio C#? .................................................. 6
      3.3.1 Add Reference to Newport XPS .NET Assembly .................................................... 6
      3.3.2 C# Code Sources ..................................................................................................... 6
   3.4 How to Use XPS .NET Assembly from LabVIEW? ............................................................. 7
      3.4.1 Add Reference to .NET Assembly ........................................................................... 7
      3.4.2 LabVIEW Code Sources ....................................................................................... 7
   3.5 How to Use XPS .NET Assembly from IronPython? .......................................................... 8
      3.5.1 Add Reference to .NET Assembly .......................................................................... 8
      3.5.2 IronPython Code Source ....................................................................................... 8
   3.6 How to Use XPS .NET Assembly from Matlab? ................................................................. 10
      3.6.1 Add Reference to .NET Assembly .......................................................................... 10
      3.6.2 Matlab Code Source ............................................................................................ 10

4.0 TCP/IP Support Functions ............................................................................................................... 11
   4.1 OpenInstrument ...................................................................................................................... 11
   4.2 CloseInstrument .................................................................................................................... 12
   4.3 SetTimeout ............................................................................................................................. 12

5.0 XPS Standard Firmware Architecture (Base Version) ................................................................. 13
   5.1 Group Definition .................................................................................................................. 13
      5.1.1 Object Structure ....................................................................................................... 13
   5.2 Positioner Definition ......................................................................................................... 14
      5.2.1 Object Structure ....................................................................................................... 14
      5.2.2 Definition of the Positions Available for Each Positioner ....................................... 15
   5.3 SingleAxis Group ............................................................................................................... 16
      5.3.1 State Diagram ......................................................................................................... 17
   5.4 Spindle Group ..................................................................................................................... 18
6.0  XPS Extended Firmware Architecture (Contact Newport) .......... 28
    6.1  SingleAxisWithClamping Group ............................................. 28
    6.1.1  State Diagram ................................................................. 29
    6.1.2  Group Clamping Sequence .................................................. 30
    6.1.2.1  Clamping State Diagram ................................................. 30
    6.2  SingleAxisTheta Group .......................................................... 31
    6.2.1  State Diagram ................................................................. 32
    6.2.2  Group Clamping Sequence .................................................. 33
    6.2.2.1  Clamping State Diagram ................................................. 33
    6.3  TZ Group ............................................................................. 34
    6.3.1  State Diagram ................................................................. 35
    6.4  User External Module Programming ........................................ 36
    6.5  ZYGO Interferometer ............................................................. 37
    6.5.1  ZMI Measurement System .................................................. 37
    6.5.2  ZYGO P2 Interface Registers ............................................... 37
    6.5.3  Status, Error, and High Nibble P2 Interface Register ............... 38
    6.5.4  ZYGO Error Code Table ..................................................... 39
    6.5.5  PEG Control Register ........................................................ 40
    6.5.6  ZYGO Axis Error Status List ............................................... 40
    6.5.7  ZYGO Axis Status List ....................................................... 41

7.0  XPS Functions Description ....................................................... 42
    7.1  Input Tests Common to all XPS Functions ................................. 42
    7.2  XPS Functions Lists ............................................................. 43
    7.2.1  Standard Functions .......................................................... 43
    7.2.1.1  CleanCoreDumpFolder ................................................... 43
    7.2.1.2  CleanTmpFolder ............................................................ 44
    7.2.1.3  CloseAllOtherSockets .................................................... 45
    7.2.1.4  ControllerMotionKernelTimeLoadGet ............................... 46
    7.2.1.5  ControllerRTTimeGet ..................................................... 47
    7.2.1.6  ControllerSlaveStatusGet ............................................... 48
    7.2.1.7  ControllerSlaveStatusGetString ....................................... 49
    7.2.1.8  ControllerStatusGet ........................................................ 50
7.2.1.60 GPIOAnalogGet ................................................................. 108
7.2.1.61 GPIOAnalogRangeConfigurationGet ................................. 109
7.2.1.62 GPIOAnalogRangeConfigurationSet ................................. 110
7.2.1.63 GPIOAnalogSet .............................................................. 111
7.2.1.64 GPIODigitalGet ............................................................... 112
7.2.1.65 GPIODigitalSet .............................................................. 113
7.2.1.66 GPIODigitalPulseWidthGet ............................................. 114
7.2.1.67 GPIODigitalPulseWidthSet ............................................. 115
7.2.1.68 GroupAccelerationCurrentGet ........................................ 116
7.2.1.69 GroupAccelerationSetpointGet ........................................ 117
7.2.1.70 GroupAnalogTrackingModeDisable .................................. 118
7.2.1.71 GroupAnalogTrackingModeEnable .................................. 119
7.2.1.72 GroupBrakeStateGet ....................................................... 121
7.2.1.73 GroupBrakeSet ............................................................... 122
7.2.1.74 GroupCorrectorOutputGet .............................................. 123
7.2.1.75 GroupCurrentFollowingErrorGet .................................... 124
7.2.1.76 GroupGantryModeGet ..................................................... 125
7.2.1.77 GroupGantryModeSet ..................................................... 126
7.2.1.78 GroupHomeSearch .......................................................... 127
7.2.1.79 GroupHomeSearchAndRelativeMove ............................... 129
7.2.1.80 GroupInitialize .............................................................. 131
7.2.1.81 GroupInitializeNoEncoderReset ....................................... 133
7.2.1.82 GroupInitializeWithEncoderCalibration ............................ 135
7.2.1.83 GroupInterlockDisable .................................................... 137
7.2.1.84 GroupInterlockEnable ..................................................... 138
7.2.1.85 GroupJogCurrentGet ....................................................... 139
7.2.1.86 GroupJogModeDisable ..................................................... 141
7.2.1.87 GroupJogModeEnable ...................................................... 142
7.2.1.88 GroupJogParametersGet .................................................. 144
7.2.1.89 GroupJogParametersSet ................................................... 146
7.2.1.90 GroupKill ..................................................................... 148
7.2.1.91 GroupMotionDisable ....................................................... 149
7.2.1.92 GroupMotionEnable ....................................................... 150
7.2.1.93 GroupMotionStatusGet .................................................. 151
7.2.1.94 GroupMoveAbort ........................................................... 152
7.2.1.95 GroupMoveAbortFast ..................................................... 154
7.2.1.96 GroupMoveAbsolute ...................................................... 156
7.2.1.97 GroupMoveEndWait ...................................................... 158
7.2.1.98 GroupMoveRelative ....................................................... 159
7.2.1.99 GroupPositionCurrentGet ............................................... 161
7.2.1.100 GroupPositionSetpointGet ............................................. 162
7.2.1.101 GroupPositionTargetGet ............................................... 163
7.2.1.102 GroupReferencingActionExecute ................................. 164
7.2.1.103 GroupReferencingStart ................................................ 166
7.2.1.104 GroupReferencingStop ................................................ 167
7.2.1.105 GroupSpinCurrentGet .................................................. 168
7.2.1.106 GroupSpinModeStop .................................................... 169
7.2.1.107 GroupSpinParametersGet ............................................. 170
7.2.1.108 GroupSpinParametersSet ............................................. 171
7.2.1.109 GroupStatusGet .......................................................... 173
7.2.1.110 GroupStatusStringGet .................................................. 174
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupVelocityCurrentGet</td>
<td>175</td>
</tr>
<tr>
<td>GroupVelocitySetpointGet</td>
<td>176</td>
</tr>
<tr>
<td>HardwareDateAndTimeGet</td>
<td>177</td>
</tr>
<tr>
<td>HardwareDateAndTimeSet</td>
<td>178</td>
</tr>
<tr>
<td>HardwareDriverAndStageGet</td>
<td>179</td>
</tr>
<tr>
<td>InstallerVersionGet</td>
<td>180</td>
</tr>
<tr>
<td>INTServitudesCommandGet</td>
<td>181</td>
</tr>
<tr>
<td>INTServitudesStatusGet</td>
<td>182</td>
</tr>
<tr>
<td>KillAll</td>
<td>183</td>
</tr>
<tr>
<td>Login</td>
<td>184</td>
</tr>
<tr>
<td>LoginS</td>
<td>185</td>
</tr>
<tr>
<td>MultipleAxesPTExecution</td>
<td>186</td>
</tr>
<tr>
<td>MultipleAxesPTLoadToMemory</td>
<td>187</td>
</tr>
<tr>
<td>MultipleAxesPTParametersGet</td>
<td>190</td>
</tr>
<tr>
<td>MultipleAxesPTPulseOutputGet</td>
<td>191</td>
</tr>
<tr>
<td>MultipleAxesPTPulseOutputSet</td>
<td>193</td>
</tr>
<tr>
<td>MultipleAxesPTResetInMemory</td>
<td>195</td>
</tr>
<tr>
<td>MultipleAxesPTVerification</td>
<td>196</td>
</tr>
<tr>
<td>MultipleAxesPTVerificationResultGet</td>
<td>198</td>
</tr>
<tr>
<td>MultipleAxesPTExecution</td>
<td>199</td>
</tr>
<tr>
<td>MultipleAxesPVTLoadToMemory</td>
<td>200</td>
</tr>
<tr>
<td>MultipleAxesPVTParametersGet</td>
<td>203</td>
</tr>
<tr>
<td>MultipleAxesPVTParametersGet</td>
<td>204</td>
</tr>
<tr>
<td>MultipleAxesPVTPulseOutputGet</td>
<td>206</td>
</tr>
<tr>
<td>MultipleAxesPVTResetInMemory</td>
<td>208</td>
</tr>
<tr>
<td>MultipleAxesPVTVerification</td>
<td>209</td>
</tr>
<tr>
<td>MultipleAxesPVTVerificationResultGet</td>
<td>211</td>
</tr>
<tr>
<td>OpenConnection</td>
<td>212</td>
</tr>
<tr>
<td>PositionerAccelerationAutoScaling</td>
<td>213</td>
</tr>
<tr>
<td>PositionerAnalogTrackingPositionParametersGet</td>
<td>215</td>
</tr>
<tr>
<td>PositionerAnalogTrackingPositionParametersSet</td>
<td>216</td>
</tr>
<tr>
<td>PositionerAnalogTrackingVelocityParametersGet</td>
<td>218</td>
</tr>
<tr>
<td>PositionerAnalogTrackingVelocityParametersSet</td>
<td>220</td>
</tr>
<tr>
<td>PositionerBacklashDisable</td>
<td>222</td>
</tr>
<tr>
<td>PositionerBacklashEnable</td>
<td>223</td>
</tr>
<tr>
<td>PositionerBacklashGet</td>
<td>225</td>
</tr>
<tr>
<td>PositionerBacklashSet</td>
<td>226</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOAbort</td>
<td>227</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOCurrentStatusGet</td>
<td>228</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOEnable</td>
<td>229</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOFromFile</td>
<td>230</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOLoadToMemory</td>
<td>232</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOMemoryReset</td>
<td>233</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOPrepare</td>
<td>234</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOPulseParametersGet</td>
<td>235</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOPulseParametersSet</td>
<td>237</td>
</tr>
<tr>
<td>PositionerCompensatedFastPCOSet</td>
<td>238</td>
</tr>
<tr>
<td>PositionerCompensatedPCOAbort</td>
<td>240</td>
</tr>
<tr>
<td>PositionerCompensatedPCOEnable</td>
<td>242</td>
</tr>
<tr>
<td>PositionerCompensatedPCOFromFile</td>
<td>244</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>PositionerCorrectorNotchFiltersGet</td>
<td>Get notch filters settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorNotchFiltersSet</td>
<td>Set notch filters settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDAccelerationFilterGet</td>
<td>Get acceleration filter settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDAccelerationFilterSet</td>
<td>Set acceleration filter settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDBaseGet</td>
<td>Get base settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDBaseSet</td>
<td>Set base settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDDualFFVoltageGet</td>
<td>Get dual FF voltage settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDDualFFVoltageSet</td>
<td>Set dual FF voltage settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFAccelerationGet</td>
<td>Get acceleration settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFAccelerationSet</td>
<td>Set acceleration settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFVelocityGet</td>
<td>Get velocity settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFVelocitySet</td>
<td>Set velocity settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIPositionGet</td>
<td>Get PI position settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIPositionSet</td>
<td>Set PI position settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPlantFeedForwardDelayGet</td>
<td>Get feedforward delay settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPlantFeedForwardDelaySet</td>
<td>Set feedforward delay settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPostFFGet</td>
<td>Get post FF settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPostFFSet</td>
<td>Set post FF settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorTypeGet</td>
<td>Get the corrector type for the positioner.</td>
</tr>
<tr>
<td>PositionerCurrentVelocityAccelerationFiltersGet</td>
<td>Get current velocity acceleration filters settings for the positioner.</td>
</tr>
<tr>
<td>PositionerCurrentVelocityAccelerationFiltersSet</td>
<td>Set current velocity acceleration filters settings for the positioner.</td>
</tr>
<tr>
<td>PositionerDriverFiltersGet</td>
<td>Get filters settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverFiltersSet</td>
<td>Set filters settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverPositionOffsetsGet</td>
<td>Get position offset settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverPositionOffsetsSet</td>
<td>Set position offset settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverStatusGet</td>
<td>Get status settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverStatusStringGet</td>
<td>Get status string for the positioner driver.</td>
</tr>
<tr>
<td>PositionerEncoderAmplitudeValuesGet</td>
<td>Get amplitude values for the encoder.</td>
</tr>
<tr>
<td>PositionerEncoderIndexDifferenceGet</td>
<td>Get index difference for the encoder.</td>
</tr>
<tr>
<td>PositionerEncoderCalibrationParametersGet</td>
<td>Get calibration parameters for the encoder.</td>
</tr>
<tr>
<td>PositionerExcitationSignalCorrectorOutSet</td>
<td>Set excitation signal correction for the excitation signal.</td>
</tr>
<tr>
<td>PositionerExcitationSignalGet</td>
<td>Get excitation signal settings for the positioner.</td>
</tr>
<tr>
<td>PositionerExcitationSignalGet</td>
<td>Get excitation signal settings for the positioner.</td>
</tr>
<tr>
<td>PositionerErrorStringGet</td>
<td>Get error string for the positioner.</td>
</tr>
<tr>
<td>PositionerErrorRead</td>
<td>Read error for the positioner.</td>
</tr>
<tr>
<td>PositionerErrorStringGet</td>
<td>Get error string for the positioner.</td>
</tr>
<tr>
<td>PositionerExcitationSignalGet</td>
<td>Get excitation signal settings for the positioner.</td>
</tr>
<tr>
<td>PositionerFeedforwardPositionSet</td>
<td>Set feedforward position settings for the positioner.</td>
</tr>
<tr>
<td>PositionerFeedforwardPositionGet</td>
<td>Get feedforward position settings for the positioner.</td>
</tr>
<tr>
<td>PositionerFeedforwardPositionEnable</td>
<td>Enable feedforward position for the positioner.</td>
</tr>
<tr>
<td>PositionerFeedforwardPositionDisable</td>
<td>Disable feedforward position for the positioner.</td>
</tr>
<tr>
<td>PositionerFeedforwardAccSet</td>
<td>Set feedforward acceleration settings for the positioner.</td>
</tr>
<tr>
<td>PositionerFeedforwardAccGet</td>
<td>Get feedforward acceleration settings for the positioner.</td>
</tr>
<tr>
<td>PositionerFeedforwardAccEnable</td>
<td>Enable feedforward acceleration for the positioner.</td>
</tr>
<tr>
<td>PositionerFeedforwardAccDisable</td>
<td>Disable feedforward acceleration for the positioner.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDDualFFVoltageSet</td>
<td>Set dual FF voltage settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDDualFFVoltageGet</td>
<td>Get dual FF voltage settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIPositionGet</td>
<td>Get PI position settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIPositionSet</td>
<td>Set PI position settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFVelocityGet</td>
<td>Get velocity settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFVelocitySet</td>
<td>Set velocity settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDBaseGet</td>
<td>Get base settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDBaseSet</td>
<td>Set base settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDAccelerationFilterGet</td>
<td>Get acceleration filter settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDAccelerationFilterSet</td>
<td>Set acceleration filter settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFAccelerationGet</td>
<td>Get acceleration settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFAccelerationSet</td>
<td>Set acceleration settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFVelocityGet</td>
<td>Get velocity settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIDFFVelocitySet</td>
<td>Set velocity settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIPositionGet</td>
<td>Get PI position settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPIPositionSet</td>
<td>Set PI position settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPlantFeedForwardDelayGet</td>
<td>Get feedforward delay settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPlantFeedForwardDelaySet</td>
<td>Set feedforward delay settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPostFFGet</td>
<td>Get post FF settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorPostFFSet</td>
<td>Set post FF settings for the positioner corrector.</td>
</tr>
<tr>
<td>PositionerCorrectorTypeGet</td>
<td>Get the corrector type for the positioner.</td>
</tr>
<tr>
<td>PositionerCurrentVelocityAccelerationFiltersGet</td>
<td>Get current velocity acceleration filters settings for the positioner.</td>
</tr>
<tr>
<td>PositionerCurrentVelocityAccelerationFiltersSet</td>
<td>Set current velocity acceleration filters settings for the positioner.</td>
</tr>
<tr>
<td>PositionerDriverFiltersGet</td>
<td>Get filters settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverFiltersSet</td>
<td>Set filters settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverPositionOffsetsGet</td>
<td>Get position offset settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverPositionOffsetsSet</td>
<td>Set position offset settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverStatusGet</td>
<td>Get status settings for the positioner driver.</td>
</tr>
<tr>
<td>PositionerDriverStatusStringGet</td>
<td>Get status string for the positioner driver.</td>
</tr>
<tr>
<td>PositionerEncoderAmplitudeValuesGet</td>
<td>Get amplitude values for the encoder.</td>
</tr>
<tr>
<td>PositionerEncoderIndexDifferenceGet</td>
<td>Get index difference for the encoder.</td>
</tr>
<tr>
<td>PositionerEncoderCalibrationParametersGet</td>
<td>Get calibration parameters for the encoder.</td>
</tr>
<tr>
<td>PositionerExcitationSignalCorrectorOutSet</td>
<td>Set excitation signal correction for the excitation signal.</td>
</tr>
<tr>
<td>PositionerExcitationSignalGet</td>
<td>Get excitation signal settings for the positioner.</td>
</tr>
<tr>
<td>PositionerExcitationSignalGet</td>
<td>Get excitation signal settings for the positioner.</td>
</tr>
</tbody>
</table>
7.2.1.264 PositionerMaximumVelocityAndAccelerationGet ...................... 403
7.2.1.265 PositionerMotionDoneGet ...................................................... 404
7.2.1.266 PositionerMotionDoneSet ......................................................... 405
7.2.1.267 PositionerMotorDualSinForceBalanceGet ................................ 406
7.2.1.268 PositionerMotorDualSinForceBalanceSet ................................ 407
7.2.1.269 PositionerPositionCompareAquadBAwaysEnable ...................... 408
7.2.1.270 PositionerPositionCompareAquadBPrescalerGet ....................... 409
7.2.1.271 PositionerPositionCompareAquadBPrescalerSet ......................... 410
7.2.1.272 PositionerPositionCompareAquadBWindowedGet ........................ 411
7.2.1.273 PositionerPositionCompareAquadBWindowedSet ........................ 412
7.2.1.274 PositionerPositionCompareDisable ........................................... 414
7.2.1.275 PositionerPositionCompareEnable ............................................. 415
7.2.1.276 PositionerPositionCompareGet ................................................. 416
7.2.1.277 PositionerPositionComparePulseParametersGet .......................... 417
7.2.1.278 PositionerPositionComparePulseParametersSet ........................... 418
7.2.1.279 PositionerPositionCompareScanAccelerationLimitGet ................ 420
7.2.1.280 PositionerPositionCompareScanAccelerationLimitSet ................ 421
7.2.1.281 PositionerPositionCompareSet .................................................. 422
7.2.1.282 PositionerPreCorrectorExcitationSignalGet ................................ 424
7.2.1.283 PositionerPreCorrectorExcitationSignalSet ............................... 425
7.2.1.284 PositionerRawEncoderPositionGet ........................................... 427
7.2.1.285 PositionerSGammaExactVelocityAdjustedDisplacementGet .......... 428
7.2.1.286 PositionerSGammaParametersGet ............................................. 429
7.2.1.287 PositionerSGammaParametersSet .............................................. 430
7.2.1.288 PositionerSGammaPreviousMotionTimesGet .............................. 432
7.2.1.289 PositionerSGammaVelocityAndAccelerationSet .......................... 433
7.2.1.290 PositionerStageParameterGet .................................................. 434
7.2.1.291 PositionerStageParameterSet .................................................... 435
7.2.1.292 PositionerTimeFlasherDisable .................................................. 436
7.2.1.293 PositionerTimeFlasherEnable ................................................... 437
7.2.1.294 PositionerTimeFlasherGet ....................................................... 438
7.2.1.295 PositionerTimeFlasherSet ....................................................... 439
7.2.1.296 PositionerUserTravelLimitsGet ................................................ 441
7.2.1.297 PositionerUserTravelLimitsSet ................................................ 442
7.2.1.298 PositionerWarningFollowingErrorGet ....................................... 443
7.2.1.299 PositionerWarningFollowingErrorSet ....................................... 444
7.2.1.300 Reboot ..................................................................................... 445
7.2.1.301 RestartApplication ..................................................................... 446
7.2.1.302 SingleAxisSlaveModeDisable .................................................... 447
7.2.1.303 SingleAxisSlaveModeEnable ..................................................... 448
7.2.1.304 SingleAxisSlaveParametersGet ................................................ 449
7.2.1.305 SingleAxisSlaveParametersSet ................................................ 450
7.2.1.306 SingleAxisThetaClampDisable .................................................. 452
7.2.1.307 SingleAxisThetaClampEnable ................................................... 453
7.2.1.308 SingleAxisThetaFeedforwardJerkParametersGet .......................... 454
7.2.1.309 SingleAxisThetaFeedforwardJerkParametersSet .......................... 455
7.2.1.310 SingleAxisThetaFeedforwardParametersGet ................................. 456
7.2.1.311 SingleAxisThetaFeedforwardParametersSet ................................. 457
7.2.1.312 SingleAxisThetaSlaveModeDisable ............................................. 458
7.2.1.313 SingleAxisThetaSlaveModeEnable .............................................. 459
7.2.1.314 SingleAxisThetaSlaveParametersGet ........................................ 460
7.2.1.315 SingleAxisThetaSlaveParametersSet ........................................... 461
7.2.1.316 SpindleSlaveModeDisable ............................................................... 463
7.2.1.317 SpindleSlaveModeEnable ............................................................... 464
7.2.1.318 SpindleSlaveParametersGet ............................................................ 465
7.2.1.319 SpindleSlaveParametersSet .............................................................. 466
7.2.1.320 TCLScriptExecute ................................................................. 468
7.2.1.321 TCLScriptExecuteAndWait .......................................................... 469
7.2.1.322 TCLScriptExecuteAndWait ......................................................... 471
7.2.1.323 TCLScriptKill ............................................. 473
7.2.1.324 TCLScriptKillAll ............................................................. 474
7.2.1.325 TCLScriptRunningListGet ......................................................... 475
7.2.1.326 TCP_CloseSocket ............................................................. 476
7.2.1.327 TCP_ConnectToServer ........................................................... 477
7.2.1.328 TCP_GetError ................................................................. 478
7.2.1.329 TCP_SetTimeout ............................................................... 479
7.2.1.330 TimerGet ................................................................. 480
7.2.1.331 TimerSet ................................................................. 481
7.2.1.332 TZEncoderCouplingMatrixGet ............................................... 482
7.2.1.333 TZEncoderCouplingMatrixSet .................................................... 484
7.2.1.334 TZEncoderCouplingModeGet ...................................................... 486
7.2.1.335 TZEncoderCouplingModeSet ...................................................... 487
7.2.1.336 TZFocusModeDisable ............................................................. 488
7.2.1.337 TZFocusModeEnable ............................................................... 489
7.2.1.338 TZMappingModeGet ............................................................... 490
7.2.1.339 TZMappingModeSet ............................................................... 491
7.2.1.340 TZMotorDecouplingMatrixGet ....................................................... 492
7.2.1.341 TZMotorDecouplingMatrixSet ...................................................... 494
7.2.1.342 TZMotorDecouplingModeGet ....................................................... 496
7.2.1.343 TZMotorDecouplingModeSet ....................................................... 497
7.2.1.344 TZPTExecution ................................................................. 498
7.2.1.345 TZPTLoadToMemory ............................................................... 500
7.2.1.346 TZPTParametersGet ............................................................... 502
7.2.1.347 TZPTPulseOutputGet ............................................................... 503
7.2.1.348 TZPTPulseOutputSet ............................................................... 505
7.2.1.349 TZPTResetInMemory ............................................................... 507
7.2.1.350 TZPTVerification ................................................................. 508
7.2.1.351 TZPTVerificationResultGet ....................................................... 510
7.2.1.352 TZPTExecution ................................................................. 512
7.2.1.353 TZPTLoadToMemory ............................................................... 514
7.2.1.354 TZPTParametersGet ............................................................... 516
7.2.1.355 TZPTPulseOutputGet ............................................................... 517
7.2.1.356 TZPTPulseOutputSet ............................................................... 519
7.2.1.357 TZPTResetInMemory ............................................................... 521
7.2.1.358 TZPTVerification ................................................................. 522
7.2.1.359 TZPTVerificationResultGet ....................................................... 524
7.2.1.360 TZTrackingCutOffFrequencyGet ................................................. 526
7.2.1.361 TZTrackingCutOffFrequencySet .................................................. 527
7.2.1.362 TZTrackingUserMaximumZZZTargetDifferenceGet ...................... 528
7.2.1.363 TZTrackingUserMaximumZZZTargetDifferenceSet ...................... 529
7.2.1.364 XYCrossTalkCompensationMotorDecouplingGet ......................... 530
7.2.1.365 XYCrossTalkCompensationMotorDecouplingSet .......................... 531
| 7.2.1.366 | XYGroupPositionCorrectedProfilerGet ................................................. 532 |
| 7.2.1.367 | XYGroupPositionPCORawEncoderGet .......................................................... 534 |
| 7.2.1.368 | XYLineArcExecution ............................................................................... 535 |
| 7.2.1.369 | XYLineArcParametersGet ......................................................................... 537 |
| 7.2.1.370 | XYLineArcPulseOutputGet ....................................................................... 538 |
| 7.2.1.371 | XYLineArcPulseOutputSet ....................................................................... 540 |
| 7.2.1.372 | XYLineArcVerification ........................................................................... 542 |
| 7.2.1.373 | XYLineArcVerificationResultGet ............................................................. 544 |
| 7.2.1.374 | XYMappingGet ........................................................................................ 545 |
| 7.2.1.375 | XYMappingSet ....................................................................................... 547 |
| 7.2.1.376 | XYPTExecution ....................................................................................... 549 |
| 7.2.1.377 | XYPTLoadToMemory ................................................................................ 551 |
| 7.2.1.378 | XYPTParametersGet ................................................................................ 553 |
| 7.2.1.379 | XYPTPulseOutputGet ............................................................................... 554 |
| 7.2.1.380 | XYPTPulseOutputSet ................................................................................ 556 |
| 7.2.1.381 | XYPTResetInMemory ................................................................................ 558 |
| 7.2.1.382 | XYPTVerification .................................................................................... 559 |
| 7.2.1.383 | XYPTVerificationResultGet ..................................................................... 561 |
| 7.2.1.384 | XYPTExecution ....................................................................................... 563 |
| 7.2.1.385 | XYPTLoadToMemory ................................................................................ 565 |
| 7.2.1.386 | XYPTParametersGet ................................................................................ 567 |
| 7.2.1.387 | XYPTPulseOutputGet ............................................................................... 568 |
| 7.2.1.388 | XYPTPulseOutputSet ................................................................................ 570 |
| 7.2.1.389 | XYPTResetInMemory ................................................................................ 572 |
| 7.2.1.390 | XYPTVerification .................................................................................... 573 |
| 7.2.1.391 | XYPTVerificationResultGet ..................................................................... 575 |
| 7.2.1.392 | XYZGroupPositionCorrectedProfilerGet ................................................... 577 |
| 7.2.1.393 | XYZGroupPositionPCORawEncoderGet ....................................................... 579 |
| 7.2.1.394 | XYZSplineExecution ................................................................................ 581 |
| 7.2.1.395 | XYZSplineParametersGet ......................................................................... 583 |
| 7.2.1.396 | XYZSplinePulseOutputGet ....................................................................... 584 |
| 7.2.1.397 | XYZSplinePulseOutputSet ....................................................................... 586 |
| 7.2.1.398 | XYZSplineVerification ........................................................................... 588 |
| 7.2.1.399 | XYZSplineVerificationResultGet ............................................................. 590 |

7.2.2  Extended Functions .................................................................................. 591
<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.2.17</td>
<td>GetZone</td>
<td>608</td>
</tr>
<tr>
<td>7.2.2.18</td>
<td>InitializeAndHomeX</td>
<td>609</td>
</tr>
<tr>
<td>7.2.2.19</td>
<td>InitializeAndHomeXY</td>
<td>611</td>
</tr>
<tr>
<td>7.2.2.20</td>
<td>InitializeAndHomeY</td>
<td>613</td>
</tr>
<tr>
<td>7.2.2.21</td>
<td>MoveAbsolute</td>
<td>615</td>
</tr>
<tr>
<td>7.2.2.22</td>
<td>MoveSlice</td>
<td>617</td>
</tr>
<tr>
<td>7.2.2.23</td>
<td>RequestType1</td>
<td>619</td>
</tr>
<tr>
<td>7.2.2.24</td>
<td>RequestType2</td>
<td>621</td>
</tr>
<tr>
<td>7.2.2.25</td>
<td>RequestType3</td>
<td>622</td>
</tr>
<tr>
<td>7.2.2.26</td>
<td>SetAccParams</td>
<td>623</td>
</tr>
<tr>
<td>7.2.2.27</td>
<td>SetBrake</td>
<td>624</td>
</tr>
<tr>
<td>7.2.2.28</td>
<td>SetGantryMode</td>
<td>625</td>
</tr>
<tr>
<td>7.2.2.29</td>
<td>SetJogAcceleration</td>
<td>627</td>
</tr>
<tr>
<td>7.2.2.30</td>
<td>SetJogVelocity</td>
<td>629</td>
</tr>
<tr>
<td>7.2.2.31</td>
<td>SetPiston</td>
<td>631</td>
</tr>
<tr>
<td>7.2.2.32</td>
<td>SetVarX</td>
<td>632</td>
</tr>
<tr>
<td>7.2.2.33</td>
<td>SetVarXSecondary</td>
<td>633</td>
</tr>
<tr>
<td>7.2.2.34</td>
<td>SetVarY</td>
<td>634</td>
</tr>
<tr>
<td>7.2.2.35</td>
<td>SetVelParams</td>
<td>635</td>
</tr>
<tr>
<td>7.2.2.36</td>
<td>SetXVelParams</td>
<td>636</td>
</tr>
<tr>
<td>7.2.2.37</td>
<td>SetYVelParams</td>
<td>637</td>
</tr>
<tr>
<td>7.2.2.38</td>
<td>SetZone</td>
<td>638</td>
</tr>
<tr>
<td>7.2.2.39</td>
<td>StartJog</td>
<td>639</td>
</tr>
<tr>
<td>7.2.2.40</td>
<td>WaitMotionEnd</td>
<td>640</td>
</tr>
<tr>
<td>7.2.2.41</td>
<td>ZygoADCDiagnosticStatusGet</td>
<td>641</td>
</tr>
<tr>
<td>7.2.2.42</td>
<td>ZygoAmplitudeGet</td>
<td>642</td>
</tr>
<tr>
<td>7.2.2.43</td>
<td>ZygoConnectToServer</td>
<td>643</td>
</tr>
<tr>
<td>7.2.2.44</td>
<td>ZygoDisconnectFromServer</td>
<td>644</td>
</tr>
<tr>
<td>7.2.2.45</td>
<td>ZygoErrorStatusGet</td>
<td>645</td>
</tr>
<tr>
<td>7.2.2.46</td>
<td>ZygoErrorStatusStringGet</td>
<td>646</td>
</tr>
<tr>
<td>7.2.2.47</td>
<td>ZygoEthernetCommunicationStatusGet</td>
<td>647</td>
</tr>
<tr>
<td>7.2.2.48</td>
<td>ZygoGetPEGLastCommunicationTime</td>
<td>648</td>
</tr>
<tr>
<td>7.2.2.49</td>
<td>ZygoGetVerInterferometer</td>
<td>649</td>
</tr>
<tr>
<td>7.2.2.50</td>
<td>ZygoInterferometerStatusGet</td>
<td>650</td>
</tr>
<tr>
<td>7.2.2.51</td>
<td>ZygoPositionGet</td>
<td>651</td>
</tr>
<tr>
<td>7.2.2.52</td>
<td>ZygoReadLong</td>
<td>652</td>
</tr>
<tr>
<td>7.2.2.53</td>
<td>ZygoReadWord</td>
<td>653</td>
</tr>
<tr>
<td>7.2.2.54</td>
<td>ZygoRegisterGet</td>
<td>654</td>
</tr>
<tr>
<td>7.2.2.55</td>
<td>ZygoRegisterSet</td>
<td>655</td>
</tr>
<tr>
<td>7.2.2.56</td>
<td>ZygoReset</td>
<td>656</td>
</tr>
<tr>
<td>7.2.2.57</td>
<td>ZygoResetX</td>
<td>657</td>
</tr>
<tr>
<td>7.2.2.58</td>
<td>ZygoResetY</td>
<td>658</td>
</tr>
<tr>
<td>7.2.2.59</td>
<td>ZygoSendAndReceive</td>
<td>660</td>
</tr>
<tr>
<td>7.2.2.60</td>
<td>ZygoSetOffSetX</td>
<td>661</td>
</tr>
<tr>
<td>7.2.2.61</td>
<td>ZygoSetOffSetY</td>
<td>662</td>
</tr>
<tr>
<td>7.2.2.62</td>
<td>ZygoSetPEGParams</td>
<td>663</td>
</tr>
<tr>
<td>7.2.2.63</td>
<td>ZygoStartBoardP2</td>
<td>665</td>
</tr>
<tr>
<td>7.2.2.64</td>
<td>ZygoStartInterferometer</td>
<td>667</td>
</tr>
<tr>
<td>7.2.2.65</td>
<td>ZygoStatusGet</td>
<td>668</td>
</tr>
<tr>
<td>7.2.2.66</td>
<td>ZygoStatusStringGet</td>
<td>669</td>
</tr>
<tr>
<td>7.2.2.67</td>
<td>ZygoWriteLong</td>
<td>670</td>
</tr>
</tbody>
</table>
8.0 Lists and Tables for XPS Functions ...................................................... 672
  8.1 Event Triggers List .......................................................................................... 672
  8.2 Actions List ........................................................................................................ 674
  8.3 Gathering Data Types ....................................................................................... 675
  8.4 External Gathering Data Types ......................................................................... 677
  8.5 Positioner Error List .......................................................................................... 678
  8.6 Positioner Hardware Status List ........................................................................ 679
  8.7 Positioner Driver Status List ............................................................................. 681
  8.8 Group Status List ............................................................................................... 682
  8.9 Error List ............................................................................................................ 685
  8.10 Controller Status List ....................................................................................... 689

9.0 Process Examples ..................................................................................... 690
  9.1 Management of Errors Example ...................................................................... 690
  9.2 Firmware Version Example .............................................................................. 691
  9.3 Gathering with Motion Example ....................................................................... 692
  9.4 External Gathering Example ............................................................................. 694
  9.5 Position Output Compare Example .................................................................. 696
  9.6 Slave-Master Mode Example ........................................................................... 698
  9.7 Jogging Example ............................................................................................... 700
  9.8 Tracking Example ............................................................................................. 702
  9.9 Backlash ........................................................................................................... 703
  9.10 Timer Event and Global Variables .................................................................. 705
  9.11 Running Several Motion Processes Simultaneously ....................................... 707

Service Form ................................................................................................. 709
1.0 Note

This XPS Programmer's Manual describes all the functions available for the XPS family of controllers.

NOTE

Not all below described functions are available for XPS standard controller. If you are interested in a function that is described but not available for your controller, please contact Newport.

NOTE

For more details on XPS features, please refer to XPS User’s Manual.
2.0 TCP/IP Communication

XPS is based on a 10/100/1000 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. The XPS allows up to 100 open sockets with 30 sockets communicating simultaneously.

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

The function call is blocked until a reply is sent by the XPS, or until the timeout value is reached. 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), in that case you cannot capture errors.

---

**NOTE**

When a communication ends with a time out fault, it could be caused by some hardware failure or the timeout value is lower than the time needed by the controller to execute the requested function.

For example, setting a timeout of one second and requesting a motion that requires 2 seconds will end with a timeout. In this situation the controller will continue the move and when accomplished will send the response back to the host.

It is the host responsibility to take the appropriate actions to handle properly these situations.
3.1 How to Install .NET Drivers for XPS Controller

3.1.1 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 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.
3.1.1 Installing the 32 bit (x86) Windows Platform

The .NET assembly is in the controller in “/Public/Drivers” folder, refer to the controller user’s manual for more details.

Download the “Newport.XPS.CommandInterface_x86.exe” to your computer.

Once downloaded, run it.

![Installation Wizard](image)

The .NET assembly “Newport.XPS.CommandInterface.dll” is installed in the GAC for x86 platforms in “C:\Windows\Microsoft.NET\assembly\GAC_32\Newport.XPS.CommandInterface\” folder and is ready for use.

3.1.2 Installing the 64 bit (x64) Windows Platform

The .NET assembly is in the controller in “/Public/Drivers” folder, refer to the controller user’s manual for more details.

Download the “Newport.XPS.CommandInterface_x64.exe” to your computer.

Once downloaded, run it.

![Installation Wizard](image)

The .NET assembly “Newport.XPS.CommandInterface.dll” is installed in the GAC for x64 platforms in “C:\Windows\Microsoft.NET\assembly\GAC_64\Newport.XPS.CommandInterface\” folder and is ready for use.
3.2 Variables Equivalent for Programming Languages

The table below describes a simple of a prototype model for different languages:

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Prototype Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td><code>int FunctionName(double inputParam, double * outputParam)</code></td>
</tr>
<tr>
<td>VBasic</td>
<td><code>Long FunctionName(ByVal SocketID As Long, ByVal inputParam As Double, outputParam As Double)</code></td>
</tr>
<tr>
<td>Matlab</td>
<td><code>[Error, outputParam] FunctionName(int32 SocketID, double inputParam)</code></td>
</tr>
<tr>
<td>Python</td>
<td><code>[Error, outputParam] FunctionName(integer SocketID, double inputParam)</code></td>
</tr>
<tr>
<td>TCL</td>
<td><code>set Error [catch &quot;FunctionName SocketID inputParam outputParam&quot;]</code></td>
</tr>
</tbody>
</table>

The table below shows parameters types needed for different languages:

<table>
<thead>
<tr>
<th>Parameter Types</th>
<th>C++</th>
<th>VBasic</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input/Output Parameters</td>
<td>bool</td>
<td>float</td>
<td>double</td>
</tr>
<tr>
<td>bool</td>
<td>bool</td>
<td>Boolean</td>
<td>bool</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>Single</td>
<td>float</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td>Double</td>
<td>double</td>
</tr>
<tr>
<td>short</td>
<td>short</td>
<td>Short</td>
<td>short</td>
</tr>
<tr>
<td>unsigned short</td>
<td>unsigned short</td>
<td>UShort</td>
<td>unsigned short</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>Integer</td>
<td>integer</td>
</tr>
<tr>
<td>unsigned int</td>
<td>unsigned int</td>
<td>UInteger</td>
<td>unsigned int</td>
</tr>
<tr>
<td>long</td>
<td>long</td>
<td>Long</td>
<td>long</td>
</tr>
<tr>
<td>unsigned long</td>
<td>unsigned long</td>
<td>ULong</td>
<td>unsigned int</td>
</tr>
<tr>
<td>long long int</td>
<td>long long int</td>
<td>Long</td>
<td>long long</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>unsigned long long</td>
<td>ULong</td>
<td>unsigned long long</td>
</tr>
<tr>
<td>char * / char[]</td>
<td>char *</td>
<td>String</td>
<td>string</td>
</tr>
<tr>
<td>charhex32</td>
<td>char *</td>
<td>String</td>
<td>string</td>
</tr>
</tbody>
</table>
3.3 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.

3.3.1 Add Reference to Newport XPS .NET Assembly

In your project add Newport.XPS.CommandInterface.dll in References from Windows GAC:

![Screenshot of Visual Studio showing references]

3.3.2 C# Code Sources

**C# Header**

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

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

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

**Create an Instance of “XPS” Object**

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

**Open XPS Connection**

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

**Call “XPS” Functions**

```csharp
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)
```

---

EDH0373En1024 — 03/18 6
Close XPS Connection
if (m_xpsInterface != null)
    m_xpsInterface.CloseInstrument();

3.4 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.

3.4.1 Add Reference to .NET Assembly
Select CommandInterfaceXPS and XPS constructor from a .Net Constructor Node (refer to Connectivity panel):

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

> Connectivity > .NET >
.Net Constructor Node

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

### 3.5 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.

#### 3.5.1 Add Reference to .NET Assembly

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

**x86:**

```python
import sys
sys.path.append(r'C:\Windows\Microsoft.NET\assembly\GAC_32\Newport.XPS.CommandInterface\v4.0_1.0.0.0__9a267756cf640dcb')
```

**x64:**

```python
import sys
sys.path.append(r'C:\Windows\Microsoft.NET\assembly\GAC_64\Newport.XPS.CommandInterface\v4.0_1.0.0.0__9a267756cf640dcb')
```

#### 3.5.2 IronPython Code Source

**IronPython Header**

```makefile
# 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()
3.6 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.

3.6.1 Add Reference to .NET Assembly

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

3.6.2 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] = myxps.GroupKill('Group1')
[code] = myxps.GroupInitialize('Group1')
[code] = myxps.GroupHomeSearch('Group1')

Close XPS Connection

% Disconnect from the XPS controller
code = myxps.CloseInstrument;
4.0  TCP/IP Support Functions

To ease the use of the TCP/IP communication, few functions have been develop by Newport. These functions uses the Microsoft System.Net.Sockets.

4.1  OpenInstrument

Name
OpenInstrument – Create and open a socket.

Description
This function is used to create and open a socket. Send Timeout is set to 1 second

Prototype
int OpenInstrument (string IP_Address, int Port, int ReceiveTimeout)

Input parameters
IP_Address string IP address of instrument.
Port int Port number.
Timeout int ReceiveTimeout in milliseconds.

Output parameters
None.

Return
• 0: No error.
• -1: Error of socket open.
4.2 CloseInstrument

**Name**
CloseInstrument – Closes the current socket.

**Description**
This function is used to close the current socket.

**Prototype**
int CloseInstrument ()

**Input parameters**
None.

**Output parameters**
None.

**Return**
- 0: No error.
- -1: Error of socket close.

4.3 SetTimeout

**Name**
SetTimeout – Configures socket timeout.

**Description**
This function is used to configure socket timeout for sending and reading.

**Prototype**
int SetTimeout (int SendingTimeout, int ReadingTimeout)

**Input parameters**
- SendingTimeout int Sending timeout in milliseconds.
- ReadingTimeout int Reading timeout in milliseconds.

**Output parameters**
None.

**Return**
- 0: No error.
- -1: Error of set timeout.
5.0 XPS Standard Firmware Architecture (Base Version)

5.1 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)

5.1.1 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.
5.2 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).

5.2.1 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:

\( \text{GroupName.PositionerName or} \)

\( 3\text{-Axis system.X-axis} \)
5.2.2 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.
5.3 \textbf{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.
5.3.1 State Diagram

Called functions:
(a) GroupInitialize
(b) GroupHomeSearch
(c) GroupMoveAbsolute
(d) GroupMoveRelative
(e) GroupSlaveModeEnable
(f) GroupMotionDisable
(g) GroupMotionEnable
(h) GroupMoveAbort
(i) GroupKill or KillAll
(j) GroupJogModeEnable
(k) GroupJogModeDisable
(l) GroupAnalogTrackingModeEnable
(m) GroupAnalogTrackingModeDisable
(n) GroupInitializeWithEncoderCalibration
(o) GroupReferencingStart
(p) GroupReferencingStop
(q) PositionerAccelerationAutoScaling
(r) PositionerCorrectorAutoTuning
(s) PositionerExcitationSignalSet / PositionerPreCorrectorExcitationSignalSet
(t) GroupSlaveModeDisable
(u) GroupInitializeNoEncoderReset
5.4 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.

![Diagram](Diagram.png)
5.4.1 State Diagram

Called functions:

(a) GroupInitialize  (l) GroupAnalogTrackingModeEnable
(b) GroupHomeSearch   (m) GroupAnalogTrackingModeDisable
(c) GroupMoveAbsolute  (n) GroupInitializeWithEncoderCalibration
(d) GroupMoveRelative  (o) GroupReferencingStart
(e) GroupMotionModeEnable (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
5.5 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.
5.5.1 State Diagram

Called functions:

(a) GroupInitialize  
(b) GroupHomeSearch  
(c) GroupMoveAbsolute  
(d) GroupMoveRelative  
(e) XYLineArcExecution  
(f) GroupMotionDisable  
(g) GroupMotionEnable  
(h) GroupMoveAbort  
(i) GroupKill or KillAll  
(j) GroupJogModeEnable

(k) GroupJogModeDisable  
(l) GroupAnalogTrackingModeEnable  
(m) GroupAnalogTrackingModeDisable  
(n) GroupInitializeWithEncoderCalibration  
(o) GroupReferencingStart  
(p) GroupReferencingStop  
(q) PositionerAccelerationAutoScaling  
(r) PositionerCorrectorAutoTuning  
(s) PositionerExcitationSignalSet / PositionerPreCorrectorExcitationSignalSet  
(t) GroupInitializeNoEncoderReset
5.6 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.
5.6.1 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
5.7 **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.
5.7.1 State Diagram

Called functions:

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

5.8.1 GPIO Name List

5.8.1.1 XPS-Q Hardware

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

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

Analog inputs
- GPIO2.ADC1: Analog Input #1 of the I/O board connector #2
- GPIO2.ADC2: Analog Input #2 of the I/O board connector #2
- GPIO2.ADC3: Analog Input #3 of the I/O board connector #2
- GPIO2.ADC4: Analog Input #4 of the I/O board connector #2

Analog outputs
- GPIO2.DAC1: Analog Output #1 of the I/O board connector #2
- GPIO2.DAC2: Analog Output #2 of the I/O board connector #2
- GPIO2.DAC3: Analog Output #3 of the I/O board connector #2
- GPIO2.DAC4: Analog Output #4 of the I/O board connector #2
### 5.8.1.2 XPS-RL or XPS-D Hardware

#### Digital inputs

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

#### Digital outputs

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

#### Analog inputs

<table>
<thead>
<tr>
<th>GPIO Board #1</th>
<th>Basic GPIO board</th>
<th>Extended GPIO board</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO Board #1</td>
<td>GPIO2.ADC</td>
<td>GPIO4.ADC</td>
</tr>
<tr>
<td>GPIO Board #2</td>
<td>GPIO22.ADC</td>
<td>GPIO42.ADC</td>
</tr>
<tr>
<td>GPIO Board #3</td>
<td>GPIO23.ADC</td>
<td>GPIO43.ADC</td>
</tr>
<tr>
<td>GPIO Board #4</td>
<td>GPIO24.ADC</td>
<td>GPIO44.ADC</td>
</tr>
</tbody>
</table>

#### Analog outputs

<table>
<thead>
<tr>
<th>GPIO Board #1</th>
<th>Basic GPIO board</th>
<th>Extended GPIO board</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO Board #1</td>
<td>GPIO2.DAC</td>
<td>GPIO4.DAC</td>
</tr>
<tr>
<td>GPIO Board #2</td>
<td>GPIO22.DAC</td>
<td>GPIO42.DAC</td>
</tr>
<tr>
<td>GPIO Board #3</td>
<td>GPIO23.DAC</td>
<td>GPIO43.DAC</td>
</tr>
<tr>
<td>GPIO Board #4</td>
<td>GPIO24.DAC</td>
<td>GPIO44.DAC</td>
</tr>
</tbody>
</table>
6.0 XPS Extended Firmware Architecture (Contact Newport)

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

6.1 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.
6.1.1 State Diagram

Called functions:

(a) GroupInitialize
(b) GroupHomeSearch
(c) GroupMoveAbsolute
(d) GroupMoveRelative
(e) GroupMotionDisable
(f) GroupMotionEnable
(g) GroupMotionAbort
(h) GroupKill or KillAll
(i) GroupJogModeEnable

(k) GroupJogModeDisable
(l) GroupAnalogTrackingModeEnable
(m) GroupAnalogTrackingModeDisable
(n) GroupInitializeWithEncoderCalibration
(o) GroupReferencingStart
(p) GroupReferencingStop
(q) PositionerAccelerationAutoScaling
(r) GroupInitializeNoEncoderReset
6.1.2 Group Clamping Sequence

6.1.2.1 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).
6.2 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).
6.2.1 State Diagram

Called functions:

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

6.2.2.1 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).
6.3 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).
6.3.1 State Diagram

Called functions:

(a) GroupInitialize
(b) GroupHomeSearch
(c) GroupMoveAbsolute
(d) GroupMoveRelative
(e) MultipleAxesPVTExecution
(f) GroupMotionDisable
(g) GroupMotionEnable
(h) GroupMoveAbort
(i) GroupKill or KillAll
(j) GroupJogModeEnable
(k) GroupJogModeDisable
(l) GroupAnalogTrackingModeEnable
(m) GroupAnalogTrackingModeDisable
(n) GroupInitializeWithEncoderCalibration
(o) GroupReferencingStart
(p) GroupReferencingStop
(q) PositionerAccelerationAutoScaling
(r) PositionerCorrectorAutoTuning
(s) PositionerExcitationSignalSet
(t) TZFocusModeEnable
(u) GroupInitializeNoEncoderReset
6.4 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:**

```plaintext
[GENERAL]

(...)

SharedLibraryModuleNames = ExternalModule1, ExternalModule2

(...)```
6.5 ZYGO Interferometer

6.5.1 ZMI Measurement System

Refer to ZMI 2400 series manual - OMP 0537_F

6.5.2 ZYGO P2 Interface Registers

<table>
<thead>
<tr>
<th>P2 Registers</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis #1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Axis status</td>
<td>Read only</td>
</tr>
<tr>
<td>2</td>
<td>Position</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Velocity</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Axis #2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Axis status</td>
<td>Read only</td>
</tr>
<tr>
<td>10</td>
<td>Position</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Velocity</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>General</td>
<td></td>
<td>Command</td>
</tr>
<tr>
<td>16</td>
<td>Board status</td>
<td></td>
</tr>
<tr>
<td>Info</td>
<td></td>
<td>Read only</td>
</tr>
<tr>
<td>18</td>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>ID</td>
<td></td>
</tr>
</tbody>
</table>
6.5.3 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

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

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.
6.5.4 ZYGO Error Code Table

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

The error description is returned by the “ErrorStringGet” function.
All error code list is returned by the “ErrorListGet” function.
6.5.5 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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>PEG Pulse Width (00 = 25 ns, 01 = 50 ns, 10 = 75 ns, 11 = 100 ns)</td>
</tr>
<tr>
<td>2</td>
<td>Enable P1→P2 operation</td>
</tr>
<tr>
<td>3</td>
<td>Enable P2→P1 operation</td>
</tr>
<tr>
<td>4</td>
<td>PEG Axis Select (0 = axis 1, 1 = axis 2)</td>
</tr>
<tr>
<td>5</td>
<td>PEG Output Enable</td>
</tr>
<tr>
<td>6</td>
<td>PEG Output Polarity (0 = Low pulse, 1 = High pulse)</td>
</tr>
<tr>
<td>7</td>
<td>Enable Delta 2</td>
</tr>
</tbody>
</table>
| 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 |

6.5.6 ZYGO Axis Error Status List

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

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.
### 6.5.7 ZYGO Axis Status List

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

The axis status register is read via **Ethernet**.
7.0 XPS Functions Description

7.1 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.
7.2 XPS Functions Lists

7.2.1 Standard Functions

7.2.1.1 CleanCoreDumpFolder

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

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if “/Admin/Public/CoreDump” exists: (-22)
- Checks 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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)

- **0:** No error.
- **-100:** Internal error.
7.2.1.3 CloseAllOtherSockets

Name
CloseAllOtherSockets – Closes all sockets beside the one used.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks “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 “Input Tests Common to all XPS 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.
7.2.1.4 ControllerMotionKernelTimeLoadGet

**Name**

ControllerMotionKernelTimeLoadGet – Gets controller motion kernel time load.

**Input tests**

- Refer to section 7.1: “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, profiler and servitudes calculation time).

\[
\text{CorrectorTimeLoad} = \frac{\text{CorrectorCalculationTime}}{\text{CorrectorISRPeriod}}
\]

\[
\text{ProfilerTimeLoad} = \frac{\text{ProfilerCalculationTime}}{\text{CorrectorISRPeriod}} \times \frac{1}{\text{ProfileGeneratorISRRatio}}
\]

\[
\text{ServitudesTimeLoad} = \frac{\text{ServitudesCalculationTime}}{\text{CorrectorISRPeriod}} \times \frac{1}{\text{ServitudesISRRatio}}
\]

\[
\text{TotalTimeLoad} = \text{CorrectorTimeLoad} + \text{ProfilerTimeLoad} + \text{ServitudesTimeLoad}
\]

**NOTE**

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

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.5 ControllerRTTimeGet

**Name**
ControllerRTTimeGet – Gets the controller corrector period and corrector calculation time.

**Input tests**
- Refer to section 7.1: “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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.6  **ControllerSlaveStatusGet**

**Name**

`ControllerSlaveStatusGet` – Gets the slave controller status code.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- **0**: No error.
7.2.1.7 **ControllerSlaveStatusStringGet**

**Name**

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

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.8 **ControllerStatusGet**

**Name**

`ControllerStatusGet` – Gets the controller status code.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

Returns the controller status code. The controller status codes are listed in section 8.10: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- **0**: No error.
7.2.1.9 **ControllerStatusRead**

**Name**

`ControllerStatusRead` – Reads the controller status code.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.10 **ControllerStatusStringGet**

**Name**

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

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.

7.2.1.11 **ControllerSynchronizeCorrectorISR**

**Name**

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

**Input tests**

- Refer to section 7.1: “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 1st, 2nd... 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.12  **DataCollectionBufferReset**

**Name**
DataCollectionBufferReset – Reset data collection buffer.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
This function clears any data in the data collection buffer.

**Prototype**
```c
int DataCollectionBufferReset(
    int SocketID,
)
```

**Input parameters**
SocketID string Socket identifier used in each function.

**Output parameters**
None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.13 **DataCollectionBufferAndTimeReset**

**Name**

DataCollectionBufferAndTimeReset – Reset data collection buffer and time stamp.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

This function clears the data collection buffer and resets the data collection time stamp. It allows to reset data and time stamp at once with a function call only.

**Prototype**

```
int DataCollectionBufferAndTimeReset(int SocketID, )
```

**Input parameters**

SocketID string Socket identifier used in each function.

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.14 **DataCollectionRequest**

**Name**

*DataCollectionRequest* – Gets data collection frame.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

This function return a frame that contains the requested first blocks (max 500) of current data collection.

The frame length can be from 0 (no block) to 500 blocks depending on the time between the successive calls of DataCollectionRequest function. If the number of available blocks is smaller than the requested blocks in the function call, the actual number of blocks will be returned without error.

**Example of a frame containing 4 data blocks**:

```
0x1234ABCD 0x1234ABCD
0x1234ABCD 0x1234ABCD
0x1234ABCD
```

**Prototype**

```c
int DataCollectionRequest(
    int SocketID,
    int NbRequestBlocks,
    int *NbReturnBlocks,
    char *Frame
)
```

**Input parameters**

- **SocketID**: string – Socket identifier used in each function.
- **NbRequestBlocks**: int – Number of data blocks to get.

**Output parameters**

- **NbReturnBlocks**: int * – Number of actually returned blocks.
- **Frame**: char * – Blocks of collected data.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- **0**: No error.
### 7.2.1.15 DataCollectionTimeStampGet

**Name**
DataCollectionTimeStampGet – Gets data collection time stamp.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
This function returns the time stamp in milliseconds that elapsed since the last reset.

**Prototype**
```c
int DataCollectionTimeStampGet(
    int SocketID,
    double *TimeStamp
)
```

**Input parameters**
- **SocketID** string Socket identifier used in each function.

**Output parameters**
- **TimeStamp** double * Data collection current time stamp (milliseconds).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.16  DataCollectionTimeStampReset

**Name**

DataCollectionTimeStampReset – Reset data collection time stamp.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

This function resets internal millisecond counter used as time stamp for data collection.

**Prototype**

```c
int DataCollectionTimeStampReset(
    int SocketID,
)
```

**Input parameters**

SocketID  
string  
Socket identifier used in each function.

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.17  DoubleGlobalArrayGet

Name
DoubleGlobalArrayGet – Gets the value of the global array of type “double”.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Verifies 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
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
7.2.1.18  **DoubleGlobalArraySet**

**Name**

`DoubleGlobalArraySet` – Sets a value for the global array of type “double”.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Verifies 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

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

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

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

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.20  ErrorStringGet

**Name**
ErrorStringGet – Gets the error description from a function error code.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
The function returns the error description corresponding to a function error code (see section 8.7: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.21  **EventActionSetAndStart**

**Name**
EventActionSetAndStart – Defines a combination of one event and one action in buffer, it launches the event and action configuration and gives an ID of the set event.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Event parameters: (-40), (-8).
- 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 event and one action, it activates the event and the action. See event specification to see which are necessary. The actions are defined in section “Events and Actions” in the XPS user’s manual.
Refer to section 8.2: “Actions List”.
EventActionSetAndStart() is useful to avoid conflict between Events/Actions configurations set from many processes that run in parallel.

---

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

**Prototype**

```c
int EventActionSetAndStart (  
    int SocketID,  
    char * ExtendedEventName,  
    char * EventParameter1,  
    char * EventParameter2,  
    char * EventParameter3,  
    char * EventParameter4  
    char * ExtendedActionName,  
    char * ActionParameter1,  
    char * ActionParameter2,  
    char * ActionParameter3,  
    char * ActionParameter4  
    int*   EventId)  
```

---

63  
EDH0373En1024 — 03/18
**Input parameters**

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

**ExtendedEventName** char *  
event name (maximum size = 250). The events are defined in section “Events and Actions” in the XPS user’s manual.

**EventParameter1** char *  
optional event’s parameter #1 (maximum size = 250).

**EventParameter2** char *  
optional event’s parameter #2 (maximum size = 250).

**EventParameter3** char *  
optional event’s parameter #3 (maximum size = 250).

**EventParameter4** char *  
optional event’s parameter #4 (maximum size = 250).

**ExtendedActionName** char *  
action full name (maximum size = 250).

**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**

**EventId** int*  
event Id

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

EventExtendedConfigurationActionGet – Gets the action combination defined in buffer.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -81: Action not configured.
7.2.1.24  **EventExtendedConfigurationActionSet**

**Name**

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

**Input tests**

- Refer to section 7.1: “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 8.2: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -32: Gathering not configured.
- -39: Mnemonic action doesn’t exist.
7.2.1.25  

**EventExtendedConfigurationTriggerGet**

**Name**

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

**Input tests**

- Refer to section 7.1: “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.

![Diagram](image)

**NOTE**

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

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0:  
  No error.
- -80:  
  Event not configured.
7.2.1.26 **EventExtendedConfigurationTriggerSet**

**Name**

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

**Input tests**

- Refer to section 7.1: “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.

![Diagram](image)

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**

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

EDH0373En1024 — 03/18
**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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

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

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

#### Input tests
- Refer to section 7.1: “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
```c
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 “Input Tests Common to all XPS 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.
7.2.1.30 **EventExtendedWait**

**Name**

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

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.31 **ExternalModuleFirmwareVersionGet**

**Name**

ExternalModuleFirmwareVersionGet – Gets the firmware version of an ExternalModule.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the module number (must >=1 and <= 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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the module number (must >=1 and <= 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the module number (must >=1 and <= *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(), ElapsedTimeGet(), ErrorStringGet(), …*) become active, while *ExternalModule* functions (*like ExternalModuleTZPositionCurrentGet(), ExternalModuleGPIODigitalInputGet(), …*) become inactive.

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

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the module number (must >=1 and <= NbExternalModules): (-17)
  Here: *NbExternalModules* is the number of modules declared in the line *SharedLibraryModuleNames* of *system.ref*
- Checks if the current socket is already reserved for another ExternalModule: (-22)
- Checks 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**

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

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>ModuleNumber</td>
<td>int</td>
<td>External module number</td>
</tr>
</tbody>
</table>

**Output parameters**

None

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

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

**Name**

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

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

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.36 **FileScriptHistoryRename**

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

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

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

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.37 FirmwareBuildVersionNumberGet

**Name**
FirmwareBuildVersionNumberGet – Gets the built firmware version

**Input tests**
- Refer to section 7.1: “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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.38 FirmwareVersionGet

**Name**

FirmwareVersionGet – Gets the version of the controller.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.39  **GatheringConfigurationGet**

**Name**

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

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
GatheringConfigurationSet – Sets a data gathering action.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 8.3: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -29: Mnemonic gathering type doesn't exist.
- -32: Gathering not configured.
- -43: Gathering running.
7.2.1.41 GatheringCurrentNumberGet

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

**Input tests**
- Refer to section 7.1: “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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -32: Gathering not configured.
7.2.1.42  GatheringDataAcquire

Name
GatheringDataAcquire – Manually Acquires one data set.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Gathering must be configured: (-32)
- Gathering must not be in progress: (-43)
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -32: Gathering not configured.
- -43: Gathering running.
- -111: Gathering buffer is full.
7.2.1.43  **GatheringDataGet**

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks index number: (-17)
  - IndexPoint ≥ 0.
  - IndexPoint < currently gathered data number.
- Checks 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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -32: Gathering not configured.
7.2.1.44 **GatheringDataMultipleLinesGet**

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks index number: (-17)
  - IndexPoint ≥ 0 (Note: index #0 = line #1)
  - IndexPoint < currently gathered data number.
- Checks 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:**

<table>
<thead>
<tr>
<th>index</th>
<th>Data1</th>
<th>Data2</th>
<th>Data3</th>
<th>Data4</th>
<th>Data5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 →</td>
<td>1</td>
<td>10</td>
<td>0.1</td>
<td>21</td>
<td>100</td>
</tr>
<tr>
<td>1 →</td>
<td>2</td>
<td>20</td>
<td>0.2</td>
<td>22</td>
<td>102</td>
</tr>
<tr>
<td>2 →</td>
<td>3</td>
<td>30</td>
<td>0.3</td>
<td>23</td>
<td>103</td>
</tr>
<tr>
<td>3 →</td>
<td>4</td>
<td>40</td>
<td>0.4</td>
<td>24</td>
<td>104</td>
</tr>
<tr>
<td>5 →</td>
<td>5</td>
<td>50</td>
<td>0.5</td>
<td>25</td>
<td>105</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>index</th>
<th>Data1</th>
<th>Data2</th>
<th>Data3</th>
<th>Data4</th>
<th>Data5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 →</td>
<td>1</td>
<td>10</td>
<td>0.1</td>
<td>21</td>
<td>100</td>
</tr>
<tr>
<td>1 →</td>
<td>2</td>
<td>20</td>
<td>0.2</td>
<td>22</td>
<td>102</td>
</tr>
<tr>
<td>2 →</td>
<td>3</td>
<td>30</td>
<td>0.3</td>
<td>23</td>
<td>103</td>
</tr>
<tr>
<td>3 →</td>
<td>4</td>
<td>40</td>
<td>0.4</td>
<td>24</td>
<td>104</td>
</tr>
<tr>
<td>5 →</td>
<td>5</td>
<td>50</td>
<td>0.5</td>
<td>25</td>
<td>105</td>
</tr>
</tbody>
</table>

“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)

<table>
<thead>
<tr>
<th>index</th>
<th>Data1</th>
<th>Data2</th>
<th>Data3</th>
<th>Data4</th>
<th>Data5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 →</td>
<td>1</td>
<td>10</td>
<td>0.1</td>
<td>21</td>
<td>100</td>
</tr>
<tr>
<td>1 →</td>
<td>2</td>
<td>20</td>
<td>0.2</td>
<td>22</td>
<td>102</td>
</tr>
<tr>
<td>2 →</td>
<td>3</td>
<td>30</td>
<td>0.3</td>
<td>23</td>
<td>103</td>
</tr>
<tr>
<td>3 →</td>
<td>4</td>
<td>40</td>
<td>0.4</td>
<td>24</td>
<td>104</td>
</tr>
<tr>
<td>5 →</td>
<td>5</td>
<td>50</td>
<td>0.5</td>
<td>25</td>
<td>105</td>
</tr>
</tbody>
</table>

“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**

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
- -32: Gathering not configured.
7.2.1.45 GatheringExternalConfigurationGet

Name
GatheringExternalConfigurationGet – Gets the current configuration of an externally triggered data gathering.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)

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

**Name**

GatheringExternalConfigurationSet – Sets an external data gathering.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 8.4: “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**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
GatheringExternalCurrentNumberGet – Gets the current and maximum number of externally gathered data points.

Input tests
- Refer to section 7.1: “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
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -32: Gathering not configured.
7.2.1.48 GatheringExternalDataGet

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks index number: (-17)
  - IndexPoint ≥ 0.
  - IndexPoint < currently gathered data number.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

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

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks number of data (>0): (-30)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -30: Gathering not started.
- -60: Error during file writing or file doesn't exist.
7.2.1.50 GatheringReset

Name
GatheringReset – Resets gathered data to start new gathering.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -43: Gathering running.
7.2.1.51  **GatheringRun**

**Name**

GatheringRun – Starts to gather data.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
GatheringRunAppend – Restarts gathering from the point it was stopped.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -32: Gathering not configured.
- -43: Gathering running.
7.2.1.53  GatheringStop

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

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks number of data (>0): (-30)
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -30: Gathering not started.
- -60: Error during file writing or file doesn't exist.
7.2.1.54 GatheringStopAndSave

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks number of data (>0): (-30)
- Checks 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 “Input Tests Common to all XPS Functions”)

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

**Name**
GetLibraryVersion – Gets 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**

```c
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.
7.2.1.56 GlobalArrayGet

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

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Verifies 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
7.2.1.57  **GlobalArraySet**

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

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Verifies 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 “Input Tests Common to all XPS Functions”)
- **0**: No error.
- **-17**: Parameter out of range or incorrect.
7.2.1.58  **GPIOAnalogGainGet**

Name

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

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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

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

Input parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>NbElements</td>
<td>int</td>
<td>Number of analog GPIO to read.</td>
</tr>
<tr>
<td>GPIONameList</td>
<td>char *</td>
<td>List of analog input names – separator is comma.</td>
</tr>
</tbody>
</table>

Output parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalogGainValueArray</td>
<td>int *</td>
<td>Value of analog input gain.</td>
</tr>
</tbody>
</table>

Return

(In addition to the results of “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks board: (-8)
- GPIO name (ADC): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if the GPIO type is DAC: (-8)

**Description**

This API returns the DAC range value.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if the GPIO type is DAC: (-8)

**Description**

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

- Input value = 0 = > Disabled
- Input value ≥12.0 V = > DAC range = 12.288 V
- Input value ∈ [10.0, 12.0] = > DAC range = 10 V
- Input value ∈ [5.0, 10.0] = > DAC range = 5 V

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
7.2.1.63 \textbf{GPIOAnalogSet}

\textbf{Name}
\textbf{GPIOAnalogSet} – Sets one or several analog outputs (DAC).

\textbf{Input tests}
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if the GPIO type is DAC: (-8)
- Checks input parameters number: (-17)

\textbf{Description}
Sets analog value array for DAC type of GPIO.

\textbf{Prototype}
\begin{verbatim}
int GPIOAnalogSet(
    int SocketID,
    int NbElements,
    char * GPIONameList,
    double * AnalogValueArray
)
\end{verbatim}

\textbf{Input parameters}
- \textbf{SocketID} int Socket identifier gets by the “TCP\_ConnectToServer” function.
- \textbf{NbElements} int Number of analog GPIO to read.
- \textbf{GPIONameList} char * List of analog GPIO names – separator is comma.
- \textbf{AnalogGainValueArray} double * Analog GPIO value array (DAC or ADC).

\textbf{Output parameters}
None.

\textbf{Return} (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -17: Not allowed action.
- -22: Not allowed action.
7.2.1.64  GPIODigitalGet

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

**Input tests**
- Refer to section 7.1: “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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -22: Not allowed action.
7.2.1.65  **GPIODigitalSet**

**Name**
GPIODigitalSet – Sets one digital output.

**Input tests**
- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -22: Not allowed action.
7.2.1.66 **GPIODigitalPulseWidthGet**

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

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.1.67  GPIODigitalPulseWidthSet

Name
GPIODigitalPulseWidthSet – Sets GPIO digital I/O pulse width (µsec).

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 (µsec).

Output parameters
None.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.1.68 **GroupAccelerationCurrentGet**

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type: (-8)
- Checks positioner name: (-18)
- Checks group name: (-19)

**Description**

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

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.69  GroupAccelerationSetpointGet

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids group name: (-19)

**Description**

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

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

*GroupAnalogTrackingModeDisable* - Exits the analog tracking mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

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

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids tracking type (“Position” or “Velocity”): (-8)
- Valids 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

*GroupBrakeStateGet* – Gets current brake command state.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if GPIO board number: (-17)
- Checks if GPIO board is present: (-100)
- Checks 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**

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

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the TCP_ConnectToServer&quot; function.</td>
</tr>
<tr>
<td>GroupName</td>
<td>char *</td>
<td>Group name (XY).</td>
</tr>
</tbody>
</table>

**Output parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrakeCommand</td>
<td>int *</td>
<td>Brake command (0 or 1).</td>
</tr>
</tbody>
</table>

**Return** (In addition to the results of “Input Tests Common to all XPS 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 …).
7.2.1.73 GroupBrakeSet

**Name**

GroupBrakeSet – Sets brake command.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks GPIO board number / Checks brake command value: (-17)
- Checks if GPIO board is present: (-100)
- Checks 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.**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
- -100: Internal error (memory allocation error …).
7.2.1.74  GroupCorrectorOutputGet

Name
GroupCorrectorOutputGet – Gets corrector output for one or all positioners of the selected group.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)
• 0: No error.
7.2.1.75  **GroupCurrentFollowingErrorGet**

**Name**

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

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids 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 “Input Tests Common to all XPS Functions”)

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

Name
GroupGantryModeGet – Gets current gantry option.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the set option mode (Option0, Option1 or Option2): (-17)
- Checks the group name is valid (must be XY group): (-19)
- Checks gantry mode getting is allowed: (-22)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.77  **GroupGantryModeSet**

**Name**

*GroupGantryModeSet* – Sets gantry option.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the user option value (Option0, Option1 or Option2): (-17)
- Checks the group name (must be XY group): (-19)
- Checks if the gantry mode getting is allowed: (-22)
- Checks the current gantry option: (-201)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.78 **GroupHomeSearch**

**Name**

`GroupHomeSearch` - Initiates a home search.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.79 GroupHomeSearchAndRelativeMove

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

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids 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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.80  GroupInitialize

Name
GroupInitialize - Initializes the motor and activates the servo loop of the selected group.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valids group name: (-19)
- Group status must be "NOTINIT": (-22)
- Checks 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

```c
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 “Input Tests Common to all XPS 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.
- -113: Both ends of run activated.
7.2.1.81 GroupInitializeNoEncoderReset

Name

GroupInitializeNoEncoderReset - Initializes the motor without encoder reset and activates the servo loop of the selected group.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valids group name: (-19)
- Group status must be "NOTINIT": (-22)
- Checks 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

```c
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 “Input Tests Common to all XPS 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.
- -113: Both ends of run activated.
7.2.1.82  **GroupInitializeWithEncoderCalibration**

**Name**

**GroupInitializeWithEncoderCalibration** - Initializes motor, calibrates encoder and activates servo loop.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valids group name: (-19)
- Group status must be “NOTINIT”: (-22)
- Checks 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

```c
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 “Input Tests Common to all XPS 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.
- -113: Both ends of run activated.
7.2.1.83  GroupInterlockDisable

Name

GroupInterlockDisable – Disables group interlock mode.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valids 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 “Input Tests Common to all XPS 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.
7.2.1.84  GroupInterlockEnable

**Name**

GroupInterlockEnable – Enables group interlock mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Values 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 “Input Tests Common to all XPS 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.
7.2.1.85  **GroupJogCurrentGet**

**Name**

*GroupJogCurrentGet* – Gets 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 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.86  GroupJogModeDisable

Name
GroupJogModeDisable – Disables the jog mode *

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group): (-8)
- Valids 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 “Input Tests Common to all XPS 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.
7.2.1.87 GroupJogModeEnable

Name
GroupJogModeEnable – Enables the jog mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group): (-8), (-18)
- Valids 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 “Input Tests Common to all XPS 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.
7.2.1.88 **GroupJogParametersGet**

**Name**

*GroupJogParametersGet* – Gets the velocity and acceleration set by “*GroupJogParametersSet*”.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids 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**

```c
int GroupJogParametersGet(
    int SocketID,
    char * GroupName,
    int NbPositioners,
    double * Velocity,
    double * Acceleration
)
```

**Input parameters**

- **SocketAddress**
  - `int SocketId`: Socket identifier gets by the “TCP_ConnectToServer” function.
- **GroupName**
  - `char * GroupName`: Group name.
- **NbPositioners**
  - `int NbPositioners`: Number of positioners in the selected group.

**Output parameters**

- **Velocity**
  - `double * Velocity`: User jog velocity array.
Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.89 GroupJogParametersSet

Name
GroupJogParametersSet – Changes the velocity and acceleration on the fly, in jog mode.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS 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.
7.2.1.90 **GroupKill**

**Name**

**GroupKill** - Kills the selected group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group): (-8), (-18)
- Valids 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 “Input Tests Common to all XPS 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.
7.2.1.91 **GroupMotionDisable**

**Name**

`GroupMotionDisable` – Disables a “READY” group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.92 **GroupMotionEnable**

**Name**

*GroupMotionEnable* – Enables a group in DISABLE state to turn the motors on and to restart corrector loops.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Actor must be a group: (-8), (-18)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.93  **GroupMotionStatusGet**

**Name**

*GroupMotionStatusGet* – Gets the motion status for one or all positioners of the selected group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.94 **GroupMoveAbort**

**Name**

*GroupMoveAbort* – aborts the motion or the jog in progress for a group or a positioner.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
**7.2.1.95 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 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids UserDecelerationMultiplier value (\( \geq 1 \) and \( \leq 100 \)): (-17)
- Valids positioner name: (-18)
- Valids 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*):

\[
\text{UserDeceleration} = \text{DecelerationMultiplier} \times \text{MaximumAcceleration}.
\]

Here: \( \text{DecelerationMultiplier} \): GroupMoveAbortFast function parameter  
\( \text{MaximumAcceleration} \): 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**

```c
int GroupMoveAbortFast(  
    int SocketID,  
    char * GroupName  
)  
```
**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>GroupName</td>
<td>char *</td>
<td>Group name.</td>
</tr>
<tr>
<td>DecelerationMultiplier</td>
<td>int</td>
<td>Braking deceleration multiplier.</td>
</tr>
</tbody>
</table>

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.96 **GroupMoveAbsolute**

**Name**

*GroupMoveAbsolute* - Initiates an absolute move for a positioner or a group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valid object type (group or positioner): (-8)
- Verifies 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 status 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 “Input Tests Common to all XPS 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.
7.2.1.97 **GroupMoveEndWait**

**Name**

GroupMoveEndWait – Wait for the end of move (XY group only).

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group is “XY”: (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- **0**: No error.
- **-8**: Wrong object type for this command.
- **-211**: Not expected position after motion.
7.2.1.98  **GroupMoveRelative**

**Name**

*GroupMoveRelative* - Initiates a relative move for a positioner or a group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Validates object type (group or positioner): (-8)
- Verifies target displacement in relation with the travel limits: (-17)
  - TargetPosition ≥ MinimumTargetPosition.
  - TargetPosition ≤ MaximumTargetPosition.
- Validates positioner name: ERR_POSITIONER_NAME (-18)
- Validates 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 status 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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.99  GroupPositionCurrentGet

Name

GroupPositionCurrentGet – Gets the current position for one or all positioners of the selected group.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids positioner name: (-18)
- Valids 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

\[
\text{int GroupPositionCurrentGet(}
\text{int SocketID,}
\text{char * GroupName,}
\text{int NbPositioners,}
\text{double * CurrentPosition}
\text{)}
\]

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 “Input Tests Common to all XPS 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.
7.2.1.100  GroupPositionSetpointGet

Name

GroupPositionSetpointGet – Gets the setpoint position for one or all positioners of the selected group.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids 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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.101  **GroupPositionTargetGet**

**Name**

*GroupPositionTargetGet* – Gets the target position for one or all positioners of the selected group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids 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 “Input Tests Common to all XPS 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.
7.2.1.102  **GroupReferencingActionExecute**

**Name**

*GroupReferencingActionExecute* – Initiates the given action, with the given sensor and parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids 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.
- Valids 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.

**Action list**

<table>
<thead>
<tr>
<th>Action list</th>
<th>Sensor to be defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>LatchOnHighToLowTransition</td>
<td>Yes</td>
</tr>
<tr>
<td>LatchOnIndex</td>
<td>None</td>
</tr>
<tr>
<td>LatchOnIndexAfterSensorHighToLowTransition</td>
<td>Yes</td>
</tr>
<tr>
<td>LatchOnLowToHighTransition</td>
<td>Yes</td>
</tr>
<tr>
<td>MoveRelative</td>
<td>None</td>
</tr>
<tr>
<td>MoveToPreviouslyLatchedPosition</td>
<td>None</td>
</tr>
<tr>
<td>SetPosition</td>
<td>None</td>
</tr>
<tr>
<td>SetPositionToHomePreset</td>
<td>None</td>
</tr>
</tbody>
</table>

**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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.103  **GroupReferencingStart**

**Name**

*GroupReferencingStart* – Starts the referencing mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
7.2.1.104  **GroupReferencingStop**

**Name**

GroupReferencingStop – Stops the referencing mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
- -35: Position is outside of travel limits.
7.2.1.105  **GroupSpinCurrentGet**

**Name**

*GroupSpinCurrentGet* – Gets the spin mode parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a Spindle group): (-8)
- Checks the positioner name: (-18)

**Description**

This function returns the current (or actual) velocity and acceleration used by the SPIN mode.

**Prototype**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.106  **GroupSpinModeStop**

**Name**

*GroupSpinModeStop* – Stops the motion of the spindle group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a Spindle group): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.107  **GroupSpinParametersGet**

**Name**

*GroupSpinParametersGet* – Gets the spin profiler parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a spindle group): (-8)
- Checks the function (must be a spindle function): (-18)

**Description**

This function returns the “Setpoint” (theoretical) velocity and acceleration used in SPIN mode.

**Prototype**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.108  **GroupSpinParametersSet**

**Name**

GroupSpinParametersSet – Sets the spin profiler parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a spindle group): (-8)
- Checks input parameter value: (-17)
  - Velocity ≤ MaximumVelocity.
  - Velocity ≥ MaximumVelocity.
  - Acceleration > 0.
  - Acceleration ≤ MaximumAcceleration.
- Checks 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 ≤ 0 = > ERROR and stop motion
- If Acceleration > MaximumAcceleration = > Acceleration = MaximumAcc.

**Prototype**

```c
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/s²).

**Output parameters**

None.
Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.109  GroupStatusGet

Name
GroupStatusGet – Gets the group status code.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids group name: (-19)

Description
Returns the group status code. The group status codes are listed in section 8.8: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.
7.2.1.110  **GroupStatusStringGet**

**Name**

*GroupStatusStringGet* – Gets the group status description from a group status code.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

This function returns the group status description corresponding to a group status code (see section 8.10: “Controller Status List”).

If the group status code is not referenced, “Error: ( “undefined status”) will be returned.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.111  **GroupVelocityCurrentGet**

**Name**

*GroupVelocityCurrentGet* – Gets the current velocity for one or all positioners of the selected group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Valids positioner name: (-18)
- Valids group name: (-19)

**Description**

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

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.112  **GroupVelocitySetpointGet**

**Name**

*GroupVelocitySetpointGet* – Gets the setpoint velocity for one or all positioners of the selected group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type: (-8)
- Checks positioner name: (-18)
- Checks group name: (-19)

**Description**

Gets the setpoint velocity for one or all positioners of the selected group.

**Prototype**

```c
int GroupVelocitySetpointGet(
    int SocketID,
    char * GroupName[250],
    int NbPositioners,
    double * SetpointVelocity
)
```

**Input parameters**

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

**Output parameters**

- **SetpointVelocity**  
  Setpoint velocity array.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.113  **HardwareDateAndTimeGet**

**Name**  
HardwareDateAndTimeGet – Gets the current date and time.

**Input tests**  
- Refer to section 7.1: “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**  

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.114  **HardwareDateAndTimeSet**

**Name**
HardwareDateAndTimeSet – Sets the date and time.

**Input tests**
- Refer to section 7.1: “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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.115 **HardwareDriverAndStageGet**

**Name**

*HardwareDriverAndStageGet* – Gets smart hardware.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks plug number: (-17)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
InstallerVersionGet– Gets the installer pack version installed in the controller.

Input tests
- Refer to section 7.1: “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

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
**7.2.1.117  INTServitudesCommandGet**

**Name**

INTServitudesCommandGet – Reads INT servitudes command.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

This function returns the INT servitudes command register value.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -126: Wrong parameter type in the command string: short or short * expected.
7.2.1.118 INTServitudesStatusGet

**Name**
INTServitudesStatusGet – Reads INT servitudes status.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
This function returns the INT servitudes status register value.

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -126: Wrong parameter type in the command string: short or short * expected.
7.2.1.119 KillAll

**Name**
KillAll – Kills all groups.

**Input tests**
- Refer to section 7.1: “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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.120 Login

Name
Login – Self-identification.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -106: Wrong user name or password.
- -123: Action not allowed, an Administrator is already logged in.
7.2.1.121  LoginS

Name
LoginS – Self-identification in secured mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the user name and the crypted 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 * CryptedPassword
)

Input parameters
Name  char * User name.
CryptedPassword  char * Crypted User password.

Output parameters
None.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -106: Wrong user name or crypted password.
- -123: Action not allowed, an Administrator is already logged in.
7.2.1.122  MultipleAxesPTExecution

Name

MultipleAxesPTExecution – Executes a PT trajectory.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length: (-3)
- Checks group type (must be a MultipleAxes group): (-8)
- Checks input value (number of executions must >0): (-17)
- Checks group name: (-19)
- Group state must be "READY": (-22)
- Checks backlash (must not be enabled): (-46)
- Checks BaseVelocity (stages.ini, must = 0): (-48)
- Checks trajectory file existence or file reading: (-61)
- Checks 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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.123 **MultipleAxesPTLoadToMemory**

**Name**

MultipleAxesPTLoadToMemory – Loads some lines of PT trajectory to the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Checks group type (must be a MultipleAxes group): (-8)
- Checks 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:**

```c
MultipleAxesPTVerification(socketId,myGroup,FromMemory)
MultipleAxesPTExecution(socketId,myGroup,FromMemory,1).
```

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.124  **MultipleAxesPTParametersGet**

**Name**

**MultipleAxesPTParametersGet**  – Gets PT trajectory parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a MultipleAxes group): (-8)
- Checks group name: (-19)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0:  No error.
- -8:  Wrong object type for this command.
- -19:  Group name doesn't exist or unknown command.
- -22:  Not allowed action.
7.2.1.125  **MultipleAxesPTPulseOutputGet**

**Name**

*MultipleAxesPTPulseOutputGet* – Gets the configuration of pulse generation for PT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a MultipleAxes group): (-8)
- Checks 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:**

```c
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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

MultipleAxesPTPulseOutputSet – Sets the configuration of pulse generation for PT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a MultipleAxes group): (-8)
- Checks group name: (-19)
- Checks 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:

<table>
<thead>
<tr>
<th>GPIO signals</th>
<th>ISA XPS controller</th>
<th>PCI XPS controller (for example XPS-RL) with basic GPIO board</th>
<th>PCI XPS controller (for example XPS-RL) with extended GPIO board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>GPIO2, pin 11</td>
<td>GPIO1.DO6</td>
<td>GPIO5.DO14</td>
</tr>
<tr>
<td>Pulses</td>
<td>GPIO2, pin 12</td>
<td>GPIO1.DO7</td>
<td>GPIO5.DO15</td>
</tr>
</tbody>
</table>

To find the GPIO connector pin number from GPIOx.DOy, refer to Appendix / General I/O Description of XPS User’s Manual.

**Example:**

MultipleAxesPTPulseOutputSet(GroupId, 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**

```c
int MultipleAxesPTPulseOutputSet(
    int SocketID,
    char GroupName[250],
    int StartElement,
    int EndElement,
    double TimeInterval
)
```
### Input parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>GroupName</td>
<td>char *</td>
<td>Group name.</td>
</tr>
<tr>
<td>StartElement</td>
<td>int</td>
<td>Start pulse element number.</td>
</tr>
<tr>
<td>EndElement</td>
<td>int</td>
<td>End pulse element number.</td>
</tr>
<tr>
<td>TimeInterval</td>
<td>double</td>
<td>Time interval between pulses (seconds).</td>
</tr>
</tbody>
</table>

### Output parameters

None.

### Return

(In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
- -22: Not allowed action.
7.2.1.127  **MultipleAxesPTResetInMemory**

**Name**

*MultipleAxesPTResetInMemory* – Deletes the content of PT trajectory buffer in the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a MultipleAxes group): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.128  **MultipleAxesPTVerification**

**Name**

*MultipleAxesPTVerification* – Checks the PT trajectory data file.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length (must ≤250): (-3)
- Checks group type (must be a MultipleAxes group): (-8)
- Checks group name: (-19)
- Checks BaseVelocity value (must = 0): (-48)
- Checks trajectory file existence and the file format: (-61)
- Checks trajectory (number of elements must >0): (-66)
- Checks velocity (Minimum Velocity ≤ Velocity ≤ Maximum Velocity): (-68)
- Checks acceleration (Minimum acc. ≤ acceleration ≤ Maximum acc.): (-69)
- Checks end output velocity (must = 0): (-70)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.129 **MultipleAxesPTVerificationResultGet**

**Name**

*MultipleAxesPTVerificationResultGet* – Gets the results of the “MultipleAxesPTVerification” function.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name length (must ≤250): (-3)
- Checks positioner name: (-18)
- Checks 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**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn’t exist or unknown command.
- -22: Not allowed action.
### MultipleAxesPVTExecution

**Name**

MultipleAxesPVTExecution – Executes a PVT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length: (-3)
- Checks group type (must be a MultipleAxes group): (-8)
- Checks input value (number of executions must >0): (-17)
- Checks group name: (-19)
- Group state must be "READY": (-22)
- Checks backlash (must not be enabled): (-46)
- Checks BaseVelocity (stages.ini, must = 0): (-48)
- Checks trajectory file existence or file reading: (-61)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.131  MultipleAxesPVTLoadToMemory

**Name**

MultipleAxesPVTLoadToMemory – Loads some lines of PVT trajectory to the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Checks group type (must be a MultipleAxes group): (-8)
- Checks 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,Vout12
…dTn,dXn1,Voutn1,dXn2,Voutn2n")
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 “Input Tests Common to all XPS 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.
7.2.1.132  MultipleAxesPVTParametersGet

**Name**  
MultipleAxesPVTParametersGet – Gets PVT trajectory parameters.

**Input tests**  
- Refer to section 7.1: "Input Tests Common to all XPS Functions".
- Checks group type (must be a MultipleAxes group): (-8)
- Checks group name: (-19)
- Checks 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**  
```c
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 "Input Tests Common to all XPS Functions")
- 0:  
  No error.
- -8:  
  Wrong object type for this command.
- -19:  
  Group name doesn't exist or unknown command.
- -22:  
  Not allowed action.
7.2.1.133  MultipleAxesPVTPulseOutputGet

**Name**

MultipleAxesPVTPulseOutputGet – Gets the configuration of pulse generation for PVT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a MultipleAxes group): (-8)
- Checks group name: (-19)

**Description**

This function returns the last configuration of pulse generation for a PVT trajectory, that was previously set by MultipleAxesPVTPulseOutputSet().

The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

**Example:**

```c
MultipleAxesPVTPulseOutputSet(MyGroup, 3, 5, 0.01)
MultipleAxesPVTPulseOutputGet(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**

```c
int MultipleAxesPVTPulseOutputGet(
    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 “Input Tests Common to all XPS Functions”)

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

Name

MultipleAxesPVTPulseOutputSet – Sets the configuration of pulse generation for PVT trajectory.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a MultipleAxes group): (-8)
- Checks group name: (-19)
- Checks 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:

<table>
<thead>
<tr>
<th>GPIO signals</th>
<th>ISA XPS controller</th>
<th>PCI XPS controller (for example XPS-RL) with basic GPIO board</th>
<th>PCI XPS controller (for example XPS-RL) with extended GPIO board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>GPIO2, pin 11</td>
<td>GPIO1.DO6</td>
<td>GPIO5.DO14</td>
</tr>
<tr>
<td>Pulses</td>
<td>GPIO2, pin 12</td>
<td>GPIO1.DO7</td>
<td>GPIO5.DO15</td>
</tr>
</tbody>
</table>

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual.

Example:

MultipleAxesPVTPulseOutputSet(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

```c
int MultipleAxesPVTPulseOutputSet(
    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 “*Input Tests Common to all XPS Functions*”)

- **0:** No error.
- **-8:** Wrong object type for this command.
- **-19:** Group name doesn't exist or unknown command.
- **-22:** Not allowed action.
7.2.1.135 **MultipleAxesPVTResetInMemory**

**Name**
MultipleAxesPVTResetInMemory – Deletes the content of the PVT trajectory buffer in the controller memory.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a MultipleAxes group): (-8)
- Checks 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**
```c
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 “Input Tests Common to all XPS 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.
7.2.1.136  **MultipleAxesPVTVerification**

**Name**

**MultipleAxesPVTVerification** – Checks the PVT trajectory data file.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length (must ≤ 250): (-3)
- Checks group type (must be a MultipleAxes group): (-8)
- Checks group name: (-19)
- Checks BaseVelocity value (must = 0): (-48)
- Checks trajectory file existence and the file format: (-61)
- Checks trajectory (number of elements must > 0): (-66)
- Checks velocity (Minimum Velocity ≤ Velocity ≤ Maximum Velocity): (-68)
- Checks acceleration (Minimum acc. ≤ acceleration ≤ Maximum acc.): (-69)
- Checks end output velocity (must = 0): (-70)
- Checks 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**

```c
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 "Input Tests Common to all XPS 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.
7.2.1.137  **MultipleAxesPVTVerificationResultGet**

**Name**

`MultipleAxesPVTVerificationResultGet` – Gets the results of the “MultipleAxesPVTVerification” function.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name length (must ≤250): (-3)
- Checks positioner name: (-18)
- Checks 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 (-22) is returned.

**Prototype**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn’t exist or unknown command.
- -22: Not allowed action.
7.2.1.138  **OpenConnection**

**Name**

OpenConnection – opens a socket to connect to TCP server (local).

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0:  No error.
7.2.1.139  PositionerAccelerationAutoScaling

Name

PositionerAccelerationAutoScaling –Executes Auto-scaling process to determine the stage scaling acceleration.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner”: (-8)
- Checks group type: (-8)
- Checks 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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.140  **PositionerAnalogTrackingPositionParametersGet**

**Name**

*PositionerAnalogTrackingPositionParametersGet* – Gets the parameters of the current tracking position mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

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

Name
PositionerAnalogTrackingPositionParametersSet – Sets the parameters of the current tracking position mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks Positioner and GPIO type (ADC): (-8)
- Checks velocity and acceleration: (-17)
- Checks 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:

\[ \text{TrackingPosition} = \text{InitialPosition} + (\text{AnalogValue} - \text{Offset}) \times \text{Scale} \]

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

```c
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 FrequecyTicks.
- **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/s²).
**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.142  **PositionerAnalogTrackingVelocityParametersGet**

**Name**

`PositionerAnalogTrackingVelocityParametersGet` – Gets the parameters of the current tracking velocity mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 tracking velocity mode.

**NOTE**

“Velocity” and “Acceleration” define the maximum velocity and acceleration used in the velocity tracking mode.

**Prototype**

```c
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 FrequecyTicks.

**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/s²).
**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
7.2.1.143  **PositionerAnalogTrackingVelocityParametersSet**

**Name**

`PositionerAnalogTrackingVelocityParametersSet` – Sets the parameters of the current tracking velocity mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks GPIO type (ADC): (-8)
- Checks Positioner: (-8), (-18)
- Checks 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:

\[
\text{InputValue} = \text{GPIOAnalogInput} - \text{Offset}
\]

\[
\text{MaxADCAmplitude} = \frac{10}{\text{GPIOAnalogGain}}
\]

if \((\text{InputValue} \geq 0)\) then

\[
\text{InputValue} = \text{InputValue} - \text{DeadBandThreshold}
\]

if \((\text{InputValue} < 0)\) then \(\text{InputValue} = 0\)
else

\[
\text{InputValue} = \text{AnalogInputValue} + \text{DeadBandThreshold}
\]

if \((\text{InputValue} > 0)\) then \(\text{InputValue} = 0\)

\[
\text{OutputValue} = (|\text{InputValue}|/\text{MaxADCAmplitude})^{\text{Order}}
\]

\[
\text{TrackingVelocity} = \text{Sign(InputValue)} * \text{OutputValue} * \text{Scale} * \text{MaxADCAmplitude}
\]

**NOTE**

The analog tracking velocity mode can be reset if the “GPIOName” parameter is blank.

**Prototype**

```c
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 FrequecyTicks.
- **GPIOName** char *: Analog input name (ADC).
- **Offset** double: Offset in volts.
- **Scale** double: Scale (Units/Volts).
- **Order** integer: Order (No unit).
- **Velocity** double: Velocity (Units/s).
- **Acceleration** double: Acceleration (Units/s²).

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
7.2.1.144  **PositionerBacklashDisable**

**Name**
PositionerBacklashDisable – Disables the backlash compensation.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.1.145  PositionerBacklashEnable

Name
PositionerBacklashEnable – Enables the backlash compensation.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Group status must be “NOTINIT”: (-22)
- Checks 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 “Input Tests Common to all XPS Functions”)

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

**Name**

*PositionerBacklashGet* – Gets the backlash compensation value.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

*PositionerBacklashSet* – Sets the backlash compensation value.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

PositionerCompensatedFastPCOAbort – Aborts the CIE fast compensated PCO pulses generation.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
7.2.1.149  **PositionerCompensatedFastPCOCCurrentStatusGet**

**Name**


**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks 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**

```c
int PositionerCompensatedFastPCOCCurrentStatusGet(  
    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 “Input Tests Common to all XPS 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.
7.2.1.150  **PositionerCompensatedFastPCOEnable**

**Name**

*PositionerCompensatedFastPCOEnable* – Activates the CIEFAST compensated PCO pulses generation.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks 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()* function, elsewhere the function fails and the error (-122) will be returned.

**Prototype**

```c
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 “Input Tests Common to all XPS 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 inexistent).
7.2.1.151 **PositionerCompensatedFastPCOFromFile**

**Name**

PositionerCompensatedFastPCOFromFile – Reads firing positions from a data file to controller’s memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks 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:

\[ \text{Position}_i \]

**Example:**

\[
\begin{align*}
\text{Position}_1 \\
\text{Position}_2 \\
\vdots \\
\text{Position}_N
\end{align*}
\]

Data conditions:

\[ \text{Position}_i > \text{Position}_{i-1}, \text{Width}_i < \text{Position}_{i+1} - \text{Position}_i. \]

**NOTE**

\( \text{Position}_i (i = 1\ldots N) \) are the offset values relative to the scanning positioner start position that is defined in the PositionerCompensatedFastPCOPrepare().

**Prototype**

```c
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 “Input Tests Common to all XPS 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 inexisten).
7.2.1.152  PositionerCompensatedFastPCOLoadToMemory

**Name**

PositionerCompensatedFastPCOLoadToMemory – Appends firing positions to controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks 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:**

```
Position, Positioni+1, Positioni+2
PositionerCompensatedLoadToMemory(XY.X, 0 0.1 0.2 0.3 0.4 0.5)
```

**Prototype**

```c
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 “Input Tests Common to all XPS 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 inexistent).
7.2.1.153  **PositionerCompensatedFastPCOMemoryReset**

**Name**

*PositionerCompensatedFastPCOMemoryReset* – Resets CIEFast compensated PCO data memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.154 PositionerCompensatedFastPCOPrepare

**Name**

PositionerCompensatedFastPCOPrepare – Prepares data for CIEFast compensated PCO pulses generation.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks direction value: (-17)
- Checks the positioner name: (-18)
- Checks if first and last PCO positions are within positions limits: (-35)
- Checks 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 1st positioner start position
-StartPosition2: Group 2nd positioner start position
-StartPosition3: Group 3rd positioner start position
- ...... etc ......

**Prototype**

```c
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 1st positioner start position (units).
StartPosition2  double  Group 2nd positioner start position (units).
StartPosition3  double  Group 3rd positioner start position (units).

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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 inexistent).
7.2.1.155 PositionerCompensatedFastPCOPulseParametersGet

Name
PositionerCompensatedFastPCOPulseParametersGet – Gets pulse configuration to compensated PCO.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks pulses generation status: (-22)
- Checks 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 *.脉冲极性
PulseToggle bool *.脉冲翻转

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.156  **PositionerCompensatedFastPCOPulseParametersSet**

**Name**

*PositionerCompensatedFastPCOPulseParametersGet* – Sets pulse configuration to compensated PCO.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks pulses generation status: (-22)
- Checks if CIEFAST compensated PCO pulses generation is supported: (-115)

**Description**

This function sets values of parameters: Pulse width, Pulse polarity and Pulse toggle.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.157  **PositionerCompensatedFastPCOSet**

**Name**

*PositionerCompensatedFastPCOSet* – Calculates a set of evenly spaced firing positions to the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks Start <Stop, Step >0, and (Stop – Start) <Step: (-17)
- Checks the positioner name: (-18)
- Checks if CIEFAST compensated PCO pulses generation is enabled: (-121)
- Checks data number \(NData = \text{floor}(\text{Stop} - \text{Start}) / \text{Step} + 1\), 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:

<table>
<thead>
<tr>
<th>Index (= 0)</th>
<th>Start</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index (= 1)</td>
<td>Start + Step</td>
</tr>
<tr>
<td>Index (= 2)</td>
<td>Start + 2 * Step</td>
</tr>
<tr>
<td>Index (= 3)</td>
<td>Start + 3 * Step</td>
</tr>
<tr>
<td>Index (= 4)</td>
<td>Start + 4 * Step</td>
</tr>
<tr>
<td>Index (= \ldots)</td>
<td>\ldots</td>
</tr>
<tr>
<td>Index (= N)</td>
<td>Start + N * Step</td>
</tr>
</tbody>
</table>

**Example:**

Send PositionerCompensatedFastPCOSet(XY.X, 0, 10, 1) will generate points table below:

<table>
<thead>
<tr>
<th>Index (= 0)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index (= 1)</td>
<td>1</td>
</tr>
<tr>
<td>Index (= 2)</td>
<td>2</td>
</tr>
<tr>
<td>Index (= 3)</td>
<td>3</td>
</tr>
<tr>
<td>Index (= 4)</td>
<td>4</td>
</tr>
<tr>
<td>Index (= \ldots)</td>
<td>\ldots</td>
</tr>
<tr>
<td>Index (= 10)</td>
<td>10</td>
</tr>
</tbody>
</table>
Prototype

```c
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 “Input Tests Common to all XPS 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 inexistent).
Name
PositionerCompensatedPCOAbort – Aborts the CIE08 compensated PCO pulses generation.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks the positioner name: (-18)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks the CIE board supports this function: (-115)
- Checks 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 (-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 “Input Tests Common to all XPS 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.
7.2.1.159  **PositionerCompensatedPCOCurrentStatusGet**

**Name**

`PositionerCompensatedPCOCurrentStatusGet` – Gets current status of CIE08 compensated PCO pulses generation.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks the positioner type: (-8)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks the CIE board supports this function: (-115)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.160  **PositionerCompensatedPCOEnable**

**Name**

*PositionerCompensatedPCOEnable* – Activates the CIE08 compensated PCO pulses generation.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks the CIE board supports this function: (-115)
- Checks CIE08CompensatedPCOMode = Enabled (system.ini): (-121)
- Checks if data have been prepared by PositionerCompensatedPCOPrepare(): (-122)
- Checks 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)
- Checks 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()* function, 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:

<table>
<thead>
<tr>
<th>Pulse settling time (µs)</th>
<th>PCO encoder frequency (kHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25</td>
</tr>
<tr>
<td>0.075</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>OK</td>
</tr>
<tr>
<td>12</td>
<td>OK</td>
</tr>
</tbody>
</table>
How to determine the PCO encoder frequency:

- For AquadB encoder:
  \[ \text{PCO encoder frequency} = \frac{\text{Velocity}}{\text{EncoderResolution}} \]

- For analog interpolated encoder:
  \[ \text{PCO encoder frequency} = \frac{\text{Velocity} \times \text{HardInterpolatorFactor}}{\text{EncoderScalePitch}} \]

**Example:** XML310 stage (EncoderScalePitch=0.004 mm, HardInterpolatorFactor=200). With ScanVelocity=10 mm/s => PCO encoder frequency = \(10 \times 200/0.004 = 500\, \text{kHz}\)

---

**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 (-24) error is returned.

---

**Prototype**

```c
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 “Input Tests Common to all XPS 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 inexistent).
7.2.1.161 **PositionerCompensatedPCOFromFile**

**Name**

PositionerCompensatedPCOFromFile – Reads firing positions from a data file to controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks the positioner type: (-8)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks the CIE board supports this function: (-115)
- Checks CIE08CompensatedPCOMode = Enabled (system.ini): (-121)
- Checks data file exists: (-61)
- Checks data from file (must be Positioni > Positioni-1, Widthi < Positioni+1 - Positioni): (-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:

```
Positioni<Space or Tabulation>Widthi<CRLF or LF>
```

**Example**:

```
Position1Width1
Position2Width2
... ...
PositionNWidthN
```

Data conditions: Positioni > Positioni-1, Widthi < Positioni+1 - Positioni

**NOTE**

Positioni (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 (-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 “Input Tests Common to all XPS 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 inexistent).
7.2.1.162  **PositionerCompensatedPCOLoadToMemory**

**Name**

*PositionerCompensatedPCOLoadToMemory* – Appends firing positions to controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks the positioner type: (-8)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks the CIE board supports this function: (-115)
- Checks CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)
- Checks 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_i<Space or Tabulation>Width_i<CRLF, LF or ;>
```

**Example :**

```
Position_1Width_1
Position_2Width_2
... ...
Position_NWidth_N
```

Or : 

```
Position_1Width_1;Position_2Width_2; ...;Position_NWidth_N
```

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$_i$ (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 (-24) error is returned.
**Prototype**

```c
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 “Input Tests Common to all XPS 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 inexistent).
7.2.1.163  PositionerCompensatedPCOMemoryReset

Name

PositionerCompensatedPCOMemoryReset – Resets CIE08 compensated PCO data memory.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks the positioner type: (-8)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks the CIE board supports this function: (-115)
- Checks 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 (-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 “Input Tests Common to all XPS 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.
7.2.1.164  PositionerCompensatedPCOPrepare

**Name**
PositionerCompensatedPCOPrepare – Prepares data for CIE08 compensated PCO pulses generation.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)
- Checks the positioner type: (-8)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks the CIE board supports this function: (-115)
- Checks CIE08CompensatedPCOMode = Enabled (system.ini): ERR_NOT_ALLOWED_MODE_DISABLED (-121)
- Checks data have been set, loaded or read from file to buffer (DataNumber must > 0 and < CIE08CompensatedPCOMaximumDataNumber (system.ini)): ERR_CHECK_DATA_INCORRECT (-122)
- Checks scanning zone exceed stage travel limits: (-35)
- Checks 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 1st positioner start position.
- StartPosition2: Group 2nd positioner start position
- StartPosition3: Group 3rd 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 (-24) error is returned.
Prototype

```c
int PositionerCompensatedPCOPrepare(
    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 1st positioner start position (units).
- **StartPosition2** double Group 2nd positioner start position (units).
- **StartPosition3** double Group 3rd positioner start position (units).

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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 inexistent).
7.2.1.165 **PositionerCompensatedPCOSet**

**Name**

*PositionerCompensatedPCOSet* – Calculates a set of evenly spaced firing positions to the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks Start < Stop, Distance > 0, Width > 0, Width < Distance, Width < Stop-Start: (-17)
- Checks the positioner name: (-18)
- Checks the position encoder ("AquadB" or "AnalogInterpolated"): (-24)
- Checks the CIE board supports this function: (-115)
- Checks CIE08CompensatedPCOMode = Enabled (*system.ini*): (-121)
- Checks 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 (-24) error is returned.

**Prototype**

```c
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 “Input Tests Common to all XPS 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 inexistent).
7.2.1.166 PositionerCompensationDisturbanceDisable

**Name**

**PositionerCompensationDisturbanceDisable** – Disables disturbance compensation for selected direction (DisabledDirection = Both, Positive or Negative)

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the filter number: (-17)

**Description**

This function disables the compensation of disturbance for the selected direction (Both, Positive or Negative).

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
7.2.1.167  **PositionerCompensationDisturbanceEnable**

**Name**

`PositionerCompensationDisturbanceEnable` – Enables disturbance compensation for selected direction (EnabledDirection = Both, Positive or Negative)

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the filter number: -17
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
PositionerCompensationDisturbanceFileLoad – Loads file to compensate disturbance in requested direction (Positive or Negative).

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the filter number: (-17)
- Checks if the disturbance compensation feature is allowed in the firmware: -205
- Checks 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 “Input Tests Common to all XPS 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
7.2.1.169  PositionerCompensationDisturbanceStatusGet

Name
PositionerCompensationDisturbanceStatusGet – Gets status of disturbance compensation in both directions.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
7.2.1.170 **PositionerCompensationDualLoopNotchFilterGet**

**Name**

*PositionerCompensationDualLoopNotchFilterGet* – Gets the notch filter parameters.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks the notch filter number: (-17)
- Checks the positioner name: (-18)
- Checks 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**

```c
int PositionerCompensationDualLoopNotchFilterGet(
    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 “Input Tests Common to all XPS 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.
7.2.1.171  PositionerCompensationDualLoopNotchFilterSet

Name
PositionerCompensationDualLoopNotchFilterSet – Sets the notch mode filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks 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 $\in [0 : 100]$
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.172 PositionerCompensationDualLoopPhaseCorrectionFilterGet

Name
PositionerCompensationDualLoopPhaseCorrectionFilterGet – Gets the phase correction filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks the notch filter number: (-17)
- Checks the positioner name: (-18)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.173  **PositionerCompensationDualLoopPhaseCorrectionFilterSet**

**Name**

*PositionerCompensationDualLoopPhaseCorrectionFilterSet* – Sets the notch mode filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks 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 ≥0
- Checks 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**

```c
int PositionerCompensationDualLoopNotchFilterSet(  
    int SocketID,  
    char * PositionerName,  
    int PhaseCorrectionFilterNumber,  
    double PhaseCorrectionFn,  
    double PhaseCorrectionFd,  
    double PhaseCorrectionGain  
)
```

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>PositionerName</td>
<td>char *</td>
<td>Positioner name.</td>
</tr>
<tr>
<td>PhaseCorrectionFilterNumber</td>
<td>int</td>
<td>Number of the selected selected phase correction filter.</td>
</tr>
<tr>
<td>PhaseCorrectionFn</td>
<td>double</td>
<td>Numerator frequency (Hertz) for phase correction filter.</td>
</tr>
<tr>
<td>PhaseCorrectionFd</td>
<td>double</td>
<td>Denominator frequency (Hertz) for phase correction filter.</td>
</tr>
<tr>
<td>PhaseCorrectionGain</td>
<td>double</td>
<td>Gain for phase correction filter.</td>
</tr>
</tbody>
</table>
Output parameters
None.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.174 **PositionerCompensationEncoderNotchFilterGet**

**Name**

*PositionerCompensationEncoderNotchFilterGet* – Gets Encoder compensation notch filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
int PositionerCompensationEncoderNotchFilterGet(
    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 5).

**Output parameters**

- **NotchFrequency** double * Frequency (Hertz) for Encoder Notch filter.
- **NotchBandwith** double * Band width (Hertz) for Encoder Notch filter.
- **NotchGain** double * Gain for Encoder Notch filter.
Return (In addition to the results of “Input Tests Common to all XPS Functions”)

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

**Name**

`PositionerCompensationEncoderNotchFilterSet` – Sets Encoder compensation notch filters parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 function sets the selected (Dual) EncoderFilter frequency notch filter parameters from the encoder compensation (F4 compensation block) configured in encoder.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

PositionerCompensationFrequencyNotchsGet – Gets pre-feedforward compensation notch filters parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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
- NotchBandwidth1
- NotchGain1
- NotchFrequency2
- NotchBandwidth2
- NotchGain2
- NotchFrequency3
- NotchBandwidth3
- NotchGain3

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

PositionerCompensationFrequencyNotchsSet – Sets pre-feedforward compensation notch filters parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks corrector type: ERR_WRONG_OBJECT_TYPE (-8)
- Checks parameter values: ERR_PARAMETER_OUT_OF_RANGE (-17)

\[
\begin{align*}
\text{NotchFrequency} & \in \left[ 0, \frac{0.5}{\text{CorrectorISRPeriod}} \right] \\
\text{NotchBandwidth} & \in \left[ 0, \frac{0.5}{\text{CorrectorISRPeriod}} \right]
\end{align*}
\]

**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

```c
int PositionerCompensationFrequencyNotchesSet(
    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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
7.2.1.178 **PositionerCompensationLowPassTwoFilterGet**

**Name**

`PositionerCompensationLowPassTwoFilterGet` – Gets the post-feedforward compensation second order low-pass filter parameters.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)

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

Name
PositionerCompensationLowPassTwoFilterSet – Sets the post-feedforward compensation second order low-pass filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks correcor type: (-8)
- Checks 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

```c
int PositionerCompensationLowPassTwoFilterSet(
    int SocketID,
    char * FullPositionerName,
    double CutOffFrequency
)
```

Input parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>FullPositionerName</td>
<td>Positioner name.</td>
</tr>
<tr>
<td>CutOffFrequency</td>
<td>Second order filter cut-off frequency (Hertz).</td>
</tr>
</tbody>
</table>

Output parameters

None.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
7.2.1.180 PositionerCompensationNotchFilterGet

Name
PositionerCompensationNotchFilterGet – Gets the notch filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks phase correction number: (-17)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.181 *PositionerCompensationNotchFilterSet*

**Name**

*PositionerCompensationNotchFilterSet* – Sets the notch filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameters value: (-17)
  - NotchFrequency \( \in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right] \)
  - NotchBandwidth \( \in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right] \)
  - NotchGain \( \in [0 : 100] \)
- Checks the positioner name: (-18)
- Checks compensation preFeed forward mode is enabled: (-24)
- Checks 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**

```c
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.
- **NotchBandwidth** double  Bandwidth (Hertz) for notch filter.
- **NotchGain** double  Gain for notch filter.

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- 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.
7.2.1.182 **PositionerCompensationNotchModeFiltersGet**

**Name**

*PositionerCompensationNotchModeFiltersGet* – Gets the post-feedforward compensation notch mode filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks corrector type: (-8)

**Description**

This function 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

*PositionerCompensationNotchModeFiltersSet* – Sets the post-feedforward compensation notch mode filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks corrector type: (-8)
- Checks parameter values: (-17)

\[
\begin{align*}
\text{NotchModeFr } & \in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right] \\
\text{NotchModeFa } & \in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right]
\end{align*}
\]

**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 “Input Tests Common to all XPS Functions”)
• 0: No error.
• -8: Wrong object type for this command.
7.2.1.184  PositionerCompensationPhaseCorrectionFilterGet

Name

PositionerCompensationPhaseCorrectionFilterGet – Gets the phase correction filter parameters.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks phase correction number: (-17)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.185  **PositionerCompensationPhaseCorrectionFilterSet**

**Name**

*PositionerCompensationPhaseCorrectionFilterSet* – Sets the notch filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameters value: (-17)
  
  • PhaseCorrectionFn \( \in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right] \)
  
  • PhaseCorrectionFd \( \in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right] \)
  
  • PhaseCorrectionGain \( \geq 0 \)
- Checks the positioner name: (-18)
- Checks 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**

```c
int PositionerCompensationPhaseCorrectionFilterSet(
    int SocketID,
    char * PositionerName,
    int PhaseCorrectionFilterNumber,
    double PhaseCorrectionFn,
    double PhaseCorrectionFd,
    double PhaseCorrectionGain
)
```

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>PositionerName</td>
<td>char *</td>
<td>Positioner name.</td>
</tr>
<tr>
<td>PhaseCorrectionFilterNumber</td>
<td>int</td>
<td>Number of the selected Notch Frequency filter.</td>
</tr>
<tr>
<td>PhaseCorrectionFn</td>
<td>double</td>
<td>Frequency (Hertz) for notch filter.</td>
</tr>
<tr>
<td>PhaseCorrectionFd</td>
<td>double</td>
<td>Band width (Hertz) for notch filter.</td>
</tr>
<tr>
<td>PhaseCorrectionGain</td>
<td>double</td>
<td>Gain for notch filter.</td>
</tr>
</tbody>
</table>
Output parameters

None.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.186  **PositionerCompensationPhaseCorrectionFiltersGet**

**Name**

*PositionerCompensationPhaseCorrectionFiltersGet* – Gets the post-feedforward compensation phase correction filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

*PositionerCompensationPhaseCorrectionFiltersSet* – Sets the post-feedforward compensation phase correction filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks corrector type: (-8)
- Checks parameter values: (-17)
  
  - PhaseCorrectionFn ∈ \[0 : \frac{0.5}{\text{CorrectorPeriod}}\]
  
  - PhaseCorrectionFd ∈ \[0 : \frac{0.5}{\text{CorrectorPeriod}}\]

**NOTE**

Refer to *system.ref* file to get CorrectorISRPeriod value.

**Description**

This function 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

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
PositionerCompensationPositionFilterGet

Name
PositionerCompensationPositionFilterGet – Gets the Position filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
7.2.1.189  **PositionerCompensationPositionFilterSet**

**Name**

*PositionerCompensationPositionFilterSet*— Sets the Position filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- **0**: No error.
- **-8**: Wrong object type for this command.
- **-17**: Parameter out of range or incorrect.
7.2.1.190  **PositionerCompensationPostExcitationFrequencyNotchFilterGet**

**Name**

`PositionerCompensationPostExcitationFrequencyNotchFilterGet` – Gets Notch filter parameters from F3 compensation block.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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.
- **NotchBandwidth** double * Band width (Hertz) for notch filter.
- **NotchGain** double * Gain for notch filter.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

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

**Name**

*PositionerCompensationPostExcitationFrequencyNotchFilterSet* – Sets Notch filter parameters from F3 compensation block.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks Notch Frequency number (1 to 10): (-17)
- Checks parameter values: (-17)
  
  - NotchFrequency \( \in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right] \) with CorrectorPeriod = 0.0001 s
    
    (10 kHz) = \( \left[ 0 : 5000 \right] \)
  
  - NotchBandwidth \( \in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right] \) with CorrectorPeriod = 0.0001 s
    
    (10 kHz) = \( \left[ 0 : 5000 \right] \)
  
  - NotchGain \( \in \left[ 0 : 100 \right] \)

**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**

```c
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.
NotchBandwith double Band width (Hertz) for notch filter.
NotchGain double Gain for notch filter.

Output parameters
None.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
7.2.1.192  PositionerCompensationPostExcitationLowPassFilterGet

Name
PositionerCompensationPostExcitationLowPassFilterGet – Gets the phase correction filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks 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 “Input Tests Common to all XPS Functions”)
• 0: No error.
• -8: Wrong object type for this command.
• -18: Positioner Name doesn't exist or unknown command.
7.2.1.193 **PositionerCompensationPostExcitationLowPassFilterSet**

**Name**

**PositionerCompensationPostExcitationLowPassFilterSet** – Sets second order low-pass filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameters value: (-17)
  - CutOffFrequency with CorrectorPeriod = 0.0001 s (10 kHz) = > [0 : 5000]
- Checks the positioner name: (-18)
- Checks compensation post excitation mode is enabled: (-24)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.194  PositionerCompensationPostExcitationNotchModeFilterGet

**Name**

`PositionerCompensationPostExcitationNotchModeFilterGet` – Gets the notch mode filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks phase correction number: (-17)
- Checks the positioner name: (-18)

**Description**

This function 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.195  **PositionerCompensationPostExcitationNotchModeFilterSet**

**Name**

*PositionerCompensationPostExcitationNotchModeFilterSet* – Sets the notch mode filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameters value: (-17)
  - NotchModeFr with CorrectorPeriod = 0.0001 s (10 kHz) = > \[0 : 5000\]
  - NotchModeFa with CorrectorPeriod = 0.0001 s (10 kHz) = > \[0 : 5000\]
- Checks the positioner name: (-18)
- Checks compensation post excitation mode is enabled: (-24)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.196 PositionerCompensationPostExcitationPhaseCorrectionFilterGet

Name

PositionerCompensationPostExcitationPhaseCorrectionFilterGet – Gets the phase correction filter parameters.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks the phase correction filter number: (-17)
- Checks 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

```c
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).
- **PhaseCorrectionFd** double *: Denominator frequency (Hertz).
- **PhaseCorrectionGain** double *: Gain for phase correction filter.
Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.197 **PositionerCompensationPostExcitationPhaseCorrectionFilterSet**

**Name**

*PositionerCompensationPostExcitationPhaseCorrectionFilterSet* – Sets the phase correction filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameters value: (-17)
  
  - PhaseCorrectionFn $\in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right]$
  
  - PhaseCorrectionFd $\in \left[ 0 : \frac{0.5}{\text{CorrectorPeriod}} \right]$
  
  - PhaseCorrectionGain $\geq 0$
- Checks the positioner name: (-18)
- Checks compensation post excitation mode is enabled: (-24)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.198  PositionerCompensationPreFeedForwardFrequencyNotchFilterGet

Name

PositionerCompensationPreFeedForwardFrequencyNotchFilterGet – Gets the notch filter parameters.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameter value: (-17)
- Checks 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 * 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.
NotchBandwith  double *  Band width (Hertz) for notch filter.
NotchGain  double *  Gain for notch filter.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.199  **PositionerCompensationPreFeedForwardFrequencyNotchFilterSet**

**Name**

`PositionerCompensationPreFeedForwardFrequencyNotchFilterSet` – Sets the notch filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks 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
- Checks the positioner name: (-18)
- Checks compensation preFeed forward mode is enabled: (-24)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.200 PositionerCompensationPreFeedForwardPhaseCorrectionFilterGet

Name
PositionerCompensationPreFeedForwardPhaseCorrectionFilterGet – Gets the phase correction filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks phase correction number: (-17)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.201  **PositionerCompensationPreFeedForwardPhaseCorrectionFilterSet**

**Name**

**PositionerCompensationPreFeedForwardPhaseCorrectionFilterSet** – Sets the phase correction filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks 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 ≥0
- Checks the positioner name: (-18)
- Checks compensation preFeed forward mode is enabled: (-24)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.202  PositionerCompensationPreFeedForwardSpatialNotchFilterGet

**Name**

*PositionerCompensationPreFeedForwardSpatialNotchFilterGet* – Gets the spatial notch filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks spatial notch number: (-17)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.203  **PositionerCompensationPreFeedForwardSpatialNotchFilterSet**

**Name**

*PositionerCompensationPreFeedForwardSpatialNotchFilterSet* – Sets the notch filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameters value: (-17)
  - If \( \text{SpatialNotchStep} > 1.0\cdot10^{-12} \)
    \[
    \frac{\text{Maximum Velocity}}{\text{SpatialNotchStep}} \in \left[ 0 \quad \frac{0.5}{\text{Corrector Period}} \right]
    \]
    (Corrector Period: see system.ref)
  - \( \text{SpatialNotchBandwidth} \in \left[ 0 \quad \frac{0.5}{\text{Corrector Period}} \right] \)
  - \( \text{SpatialNotchGain} \in [0:100] \)
  - Else
    - \( \text{SpatialNotchStep} = 0 \)
    - \( \text{SpatialNotchBandwidth} = 0 \)
    - \( \text{SpatialNotchGain} = 1 \)
- Checks the positioner name: (-18)
- Checks compensation preFeed forward mode is enabled: (-24)
- Checks 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**

\[
\text{int PositionerCompensationPreFeedForwardSpatialNotchFilterSet(} \\
\text{  int SocketID,} \\
\text{  char * PositionerName,} \\
\text{  int SpatialNotchNumber,} \\
\text{  double SpatialNotchStep,} \\
\text{  double SpatialNotchBandwidth,} \\
\text{  double SpatialNotchGain)} \\
\text{)}
\]
**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 “Input Tests Common to all XPS 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.
7.2.1.204  PositionerCompensationSpatialPeriodicNotchsGet

Name
PositionerCompensationSpatialPeriodicNotchsGet – Gets pre-feedforward compensation spatial periodic filters parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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

```c
int PositionerCompensationSpatialPeriodicNotchsGet(
    int SocketID,
    char * FullPositionerName,
    double * SpatialNotchStep1,
    double * SpatialNotchsBandwidth1,
    double * SpatialNotchsGain1,
    double * SpatialNotchStep2,
    double * SpatialNotchsBandwidth2,
    double * SpatialNotchsGain2,
    double * SpatialNotchStep3,
    double * SpatialNotchsBandwidth3,
    double * SpatialNotchsGain3
)
```

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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.1.205  PositionerCompensationSpatialPeriodicNotchsSet

**Name**

PositionerCompensationSpatialPeriodicNotchsSet – Sets pre-feedforward compensation spatial periodic filters parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks corrector type: (-8)
- Checks parameter values: (-17)
  - \( \text{SpatialNotchStep} \in \left[ 0 : \frac{\text{MaximumVelocity} \times \text{CorrectorISRPeriod}}{0.5 \times \text{CorrectorISRPeriod}} \right] \)
  - \( \text{SpatialNotchBandwidth} \in \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right] \)

---

**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.
- SpatialNotchBandwidth1.
- SpatialNotchGain1.
- SpatialNotchStep2.
- SpatialNotchBandwidth2.
- SpatialNotchGain2.
- SpatialNotchStep3.
- SpatialNotchBandwidth3.
- SpatialNotchGain3.
Prototype

```c
int PositionerCompensationSpatialPeriodicNotchesSet(
    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 “Input Tests Common to all XPS Functions”)

- **0**: No error.
- **-8**: Wrong object type for this command.
- **-17**: Parameter out of range or incorrect.
7.2.1.206  

**PositionerCorrectorAutoTuning**

**Name**

PositionerCorrectorAutoTuning – Executes auto-tuning process for determining position control loop PID values.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type: (-8)
- Positioner must not be a “Secondary Positioner”: (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
- -103: Relay Feedback Test failed: Signal data not enough for analyse.
- -104: Error of tuning process initialization.
7.2.1.207 PositionerCorrectorDamperFilterGet

Name
PositionerCorrectorDamperFilterGet – Gets Dual control loop parameters for a selected positioner.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.208 PositionerCorrectorDamperFilterSet

**Name**

PositionerCorrectorDamperFilterSet – Sets Dual control loop parameters for a selected positioner.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameters value: (-17)
- CutOffFrequency ≥0 and ≤(0.5 / ISRCorrectorPeriod)
- DamperFactor >0
- Gain ≤0
- Checks the motion status (Motion status must be disable): (-134)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.209 **PositionerCorrectorDualGet**

**Name**

`PositionerCorrectorDualGet` – Gets Dual control loop parameters for a selected positioner.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks the positioner name: (-18)
- Checks 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**

```c
int PositionerCorrectorDualGet(
    int SocketID,
    char * PositionerName,
    bool * ClosedLoopStatus,
    double * KP,
    double * KI,
    double * KD,
    double * IntegrationTime,
    double * DerivativeFilterCutOffFrequency,
    double * KFeedForwardAcceleration,
    double * KFeedForwardJerk
);
```

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>PositionerName</td>
<td>char *</td>
<td>Positioner name.</td>
</tr>
</tbody>
</table>
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 “Input Tests Common to all XPS 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.
7.2.1.210  **PositionerCorrectorDualSet**

**Name**

*PositionerCorrectorDualSet* – Sets Dual control loop parameters for a selected positioner.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks input parameters value: (-17)
  - KP ≥0, KI ≥0, KD ≥0, KFeedForwardAcceleration ≥0, KFeedForwardJerk ≥0
  - IntegrationTime ≥CorrectorPeriod (0.0001 s)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.211  **PositionerCorrectorExcitationSignalGainGet**

**Name**

*PositionerCorrectorExcitationSignalGainGet* – Gets corrector excitation signal gain.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)

**Description**

This function returns the corrector excitation signal gain for the selected positioner.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- **0**: No error.
- **-18**: Positioner Name doesn't exist or unknown command.
7.2.1.212  PositionerCorrectorExcitationSignalGainSet

**Name**
PositionerCorrectorExcitationSignalGainSet  – Sets corrector excitation signal gain.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the parameter value: value must be between -1 and 1, else return “Parameter out of range or incorrect” (-17)
- Checks the positioner name: (-18)

**Description**
This function configures the corrector excitation signal gain for the selected positioner. ExcitationSignalGain value must ≥-1 and ≤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 “Input Tests Common to all XPS 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.
7.2.1.213  PositionerCorrectorNotchFiltersGet

**Name**
PositionerCorrectorNotchFiltersGet – Gets the notch filter parameters.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**
```c
int PositionerCorrectorNotchFiltersGet(
    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.
**Output parameters**

- 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.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -24: Not available in this configuration (check hardware or configuration).
7.2.1.214  PositionerCorrectorNotchFiltersSet

Name
PositionerCorrectorNotchFiltersSet – Sets the notch filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks parameter values: (-17)
  - NotchFrequency $\in \left[ 0; \frac{0.5}{\text{CorrectorISRPeriod}} \right]$
  - NotchBandwidth $\in \left[ 0; \frac{0.5}{\text{CorrectorISRPeriod}} \right]$
  - 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

```c
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 “Input Tests Common to all XPS 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).
7.2.1.215  **PositionerCorrectorPIDAccelerationFilterGet**

**Name**

**PositionerCorrectorPIDAccelerationFilterGet** – Gets PID acceleration corrector filter parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)

**Description**

This function returns PID acceleration corrector filter parameters.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
PositionerCorrectorPIDAccelerationFilterSet – Sets PID acceleration corrector filter parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks the motion status (Motion status must be disable): (-134)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.217  PositionerCorrectorPIDBaseGet

**Name**

PositionerCorrectorPIDBaseGet – Gets PIDBase parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name: (-18)

**Description**

This function returns the PIDBase parameter values.

- PIDBase parameters:
  - MovingMass.
  - StaticMass.
  - Viscosity.
  - Stiffness.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- **0:** No error.
- **-8:** Wrong object type for this command.
- **-18:** Positioner Name doesn't exist or unknown command.
7.2.1.218  **PositionerCorrectorPIDBaseSet**

**Name**

*PositionerCorrectorPIDBaseSet* – Sets PIDBase parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the parameter values: all values must ≥0, else return “Parameter out of range or incorrect” (-17)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
### 7.2.1.219 PositionerCorrectorPIDDualFFVoltageGet

#### Name

**PositionerCorrectorPIDDualFFVoltageGet** – Gets PIDDualFFVoltage corrector parameters.

#### Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClosedLoopStatus</td>
<td>bool *</td>
<td>Position servo loop status (true = closed and false = opened).</td>
</tr>
<tr>
<td>KP</td>
<td>double *</td>
<td>PID servo loop proportional gain.</td>
</tr>
<tr>
<td>KI</td>
<td>double *</td>
<td>PID servo loop integral gain.</td>
</tr>
<tr>
<td>KD</td>
<td>double *</td>
<td>PID servo loop derivative gain.</td>
</tr>
<tr>
<td>KS</td>
<td>double *</td>
<td>PID integral saturation value (0 to 1).</td>
</tr>
<tr>
<td>IntegrationTime</td>
<td>double *</td>
<td>PID integration time (seconds).</td>
</tr>
<tr>
<td>DerivativeFilterCutOffFrequency</td>
<td>double *</td>
<td>PID derivative filter cut-off frequency (Hz).</td>
</tr>
<tr>
<td>GKP</td>
<td>double *</td>
<td>Variable PID proportional gain multiplier.</td>
</tr>
<tr>
<td>GKI</td>
<td>double *</td>
<td>Variable PID integral gain multiplier.</td>
</tr>
<tr>
<td>GKD</td>
<td>double *</td>
<td>Variable PID derivative gain multiplier.</td>
</tr>
<tr>
<td>KForm</td>
<td>double *</td>
<td>Variable PID form coefficient.</td>
</tr>
<tr>
<td>FeedForwardGainVelocity</td>
<td>double *</td>
<td>Velocity feedforward gain (units).</td>
</tr>
<tr>
<td>FeedForwardGainAcceleration</td>
<td>double *</td>
<td>Acceleration feedforward gain (units).</td>
</tr>
<tr>
<td>Friction</td>
<td>double *</td>
<td>Friction compensation.</td>
</tr>
</tbody>
</table>

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

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

Name

PositionerCorrectorPIDDualFFVoltageSet – Sets PIDDualFFVoltage corrector parameters.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type and the corrector type: (-8)
- Checks parameter value: (-17)
  - KP ≥0.
  - KI ≥0.
  - KD ≥0.
  - 0 ≤KS ≤1.
  - IntegrationTime ≥CorrectorISRPeriod.
  - GKP >-1.
  - GKI >-1.
  - GKD >-1.
  - KForm ≥0.
  - KFeedForwardVelocity ≥0.
  - KFeedForwardAcceleration ≥0.
  - Friction ≥0.
  - DerivativeFilterCutOffFrequency ∈ [0: \( \frac{0.5}{\text{CorrectorISRPeriod}} \)]

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

```c
int PositionerCorrectorPIDDualFFVoltageSet(
    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.
- **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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
PositionerCorrectorPIDFFAccelerationGet

Name
PositionerCorrectorPIDFFAccelerationGet – Gets PIDFFAcceleration corrector parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 * Acceleration feedforward gain (units).
FeedForwardGainJerk double * Jerk feedforward gain (units).

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
• 0: No error.
• -8: Wrong object type for this command.
7.2.1.222  PositionerCorrectorPIDFAccelerationSet

Name

PositionerCorrectorPIDFAccelerationSet – Sets PIDFAcceleration corrector parameters.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type and the corrector type: (-8)
- Checks parameter value: (-17)
  - KP ≥0.
  - KI ≥0.
  - KD ≥0.
  - 0 ≤KS ≤1.
  - IntegrationTime ≥CorrectorISRPeriod.
  - GKP >-1.
  - GKI >-1.
  - GKD >-1.
  - KForm ≥0.
  - KFeedForwardAcceleration ≥0.
  - KFeedForwardJerk ≥0.
  - DerivativeFilterCutOffFrequency ≥ \left[ 0 : \frac{0.5}{CorrectorISRPeriod} \right]

NOTE

Refer to system.ref file to get CorrectorISRPeriod value.

Description

This function configures the “PIDFAcceleration” corrector parameters.

NOTE

The “CorrectorType” parameter must be defined as “PIDFAcceleration” 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

```c
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 
  Acceleration feedforward gain (units).

- **FeedForwardGainJerk** double 
  Jerk feedforward gain (units).

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- **0:** No error.
- **-8:** Wrong object type for this command.
- **-17:** Parameter out of range or incorrect.
7.2.1.223  PositionerCorrectorPIDFFVelocityGet

Name
PositionerCorrectorPIDFFVelocityGet – Gets PIDFFVelocity corrector parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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
```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**
PositionerCorrectorPIDFFVelocitySet – Sets PIDFFVelocity corrector parameters.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type and the corrector type: (-8)
- Checks parameter value: (-17)
  - KP ≥0.
  - KI ≥0.
  - KD ≥0.
  - 0 ≤KS ≤1.
  - IntegrationTime ≥CorrectorISRPeriod.
  - GKP >-1.
  - GKI >-1.
  - GKD >-1.
  - KForm ≥0.
  - KFeedForwardVelocity ≥0.
  - DerivativeFilterCutOffFrequency ∈ \( \left[ 0 : \frac{0.5}{\text{CorrectorISRPeriod}} \right] \)

**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**

```c
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**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>FullPositionerName</td>
<td>char *</td>
<td>Positioner name.</td>
</tr>
<tr>
<td>ClosedLoopStatus</td>
<td>bool</td>
<td>Position servo loop status (true = closed and false = opened).</td>
</tr>
<tr>
<td>KP</td>
<td>double</td>
<td>PID servo loop proportional gain.</td>
</tr>
<tr>
<td>KI</td>
<td>double</td>
<td>PID servo loop integral gain.</td>
</tr>
<tr>
<td>KD</td>
<td>double</td>
<td>PID servo loop derivative gain.</td>
</tr>
<tr>
<td>KS</td>
<td>double</td>
<td>PID integral saturation value (0 to 1).</td>
</tr>
<tr>
<td>IntegrationTime</td>
<td>double</td>
<td>PID integration time (seconds).</td>
</tr>
<tr>
<td>DerivativeFilterCutOffFrequency</td>
<td>double</td>
<td>PID derivative filter cut-off frequency (Hz).</td>
</tr>
<tr>
<td>GKP</td>
<td>double</td>
<td>Variable PID proportional gain multiplier.</td>
</tr>
<tr>
<td>GKI</td>
<td>double</td>
<td>Variable PID integral gain multiplier.</td>
</tr>
<tr>
<td>GKD</td>
<td>double</td>
<td>Variable PID derivative gain multiplier.</td>
</tr>
<tr>
<td>KForm</td>
<td>double</td>
<td>Variable PID form coefficient.</td>
</tr>
<tr>
<td>FeedForwardGainVelocity</td>
<td>double</td>
<td>Velocity feedforward gain (units).</td>
</tr>
</tbody>
</table>

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- **0**: No error.
- **-8**: Wrong object type for this command.
- **-17**: Parameter out of range or incorrect.
7.2.1.225  PositionerCorrectorPIPositionGet

Name
PositionerCorrectorPIPositionGet – Gets PIPosition corrector parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 * PI servo loop proportional gain.
KI double * PI servo loop integral gain.
IntegrationTime double * PI integration time (seconds).

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.1.226  **PositionerCorrectorPIPositionSet**

**Name**

`PositionerCorrectorPIPositionSet` – Sets PIPosition corrector parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type and the corrector type: (-8)
- Checks parameter value: (-17)
- KP ≥0.
- KI ≥0.
- IntegrationTime ≥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**

```cpp
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 PI servo loop proportional gain.
- **KI** double PI servo loop integral gain.
- **IntegrationTime** double PI integration time (seconds).
Output parameters

None.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)

• 0: No error.
• -8: Wrong object type for this command.
• -17: Parameter out of range or incorrect.
7.2.1.227  PositionerCorrectorPlantFeedForwardDelayGet

Name

PositionerCorrectorPlantFeedForwardDelayGet – Gets the corrector ISR period number configured to define the plant delay.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)

- 0:  No error.
- -8:  Wrong object type for this command.
- -17: Parameter out of range or incorrect.
7.2.1.228  PositionerCorrectorPlantFeedForwardDelaySet

**Name**

PositionerCorrectorPlantFeedForwardDelaySet – Sets the corrector ISR period number that defines the plant delay.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the filter number: (-17)

**Description**

This function reconfigures the corrector ISR period number to define the plant delay. Its value is predifined in stages.ini with the parameter “PlantFeedForwardDelayCorrectorISRPeriodNumber”.

![Diagram of FeedForward and PID with plant without delay response and plant with delay response](image)

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)  
- 0: No error.  
- -8: Wrong object type for this command.  
- -17: Parameter out of range or incorrect.
7.2.1.229  PositionerCorrectorPostFFGet

**Name**
PositionerCorrectorPostFFGet – Gets Post Feed Forward parameters of a PIDFFAccelection corrector.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name: (-18)

**Description**
This function returns the parameters of the current post feed forward from PIDFFAccelection corrector.

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -18: Positioner name doesn’t exist or incorrect.
7.2.1.230  PositionerCorrectorPostFFSet

**Name**

PositionerCorrectorPostFFSet – Sets Post Feed Forward parameters of a PIDFFAcceleration corrector.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name: (-18)
- Not allowed due to configuration disabled: (-121)

**Description**

This function sets the parameters of the Post Feed Forward from PIDFFAcceleration corrector.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -18: Positioner name doesn’t exist or incorrect.
- -121: Function is not allowed due to configuration disabled.
7.2.1.231  **PositionerCorrectorTypeGet**

**Name**

*PositionerCorrectorTypeGet* – Gets the corrector type.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

PositionerCurrentVelocityAccelerationFiltersGet – Gets the velocity and acceleration filter cut-off frequencies.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

*PositionerCurrentVelocityAccelerationFiltersSet* – Sets the velocity and acceleration filter cut-off frequencies.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- **0:** No error.
- **-8:** Wrong object type for this command.
- **-17:** Parameter out of range or incorrect.
7.2.1.234  PositionerDriverFiltersGet

**Name**

PositionerDriverFiltersGet – Gets the piezo driver notch and low-pass filters parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks 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, low-pass frequency).

**Prototype**

```c
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 low-pass frequency (Hz).

**Return**  
(In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.235  **PositionerDriverFiltersSet**

**Name**

*PositionerDriverFiltersSet* – Sets the piezo driver filters parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks parameter value: (-17)
  - $\text{KI} \geq 0$.
  - $\text{NotchFrequency} \in \left[0, \frac{0.5}{\text{Corrector ISR Period}}\right]$.
  - $\text{NotchBandwidth} \in \left[0, \frac{0.5}{\text{Corrector ISR Period}}\right]$.
  - $\text{NotchGain} \in [0:100]$.
  - $\text{LowpassFrequency} \in \left[0, \frac{0.5}{\text{Corrector ISR Period}}\right]$.
- Checks 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 low-pass filters) for a piezo driver positioner.

**Prototype**

```c
int PositionerDriverFiltersSet(  
    int SocketID,  
    char * FullPositionerName,  
    double KI,  
    double NotchFrequency,  
    double NotchBandwidth,  
    double NotchGain,  
    double LowpassFrequency  
)
```
### Input parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>FullPositionerName</td>
<td>char *</td>
<td>Positioner name.</td>
</tr>
<tr>
<td>KI</td>
<td>double</td>
<td>Driver KI.</td>
</tr>
<tr>
<td>NotchFrequency</td>
<td>double</td>
<td>Driver notch frequency (Hz).</td>
</tr>
<tr>
<td>NotchBandwidth</td>
<td>double</td>
<td>Driver notch bandwidth (Hz).</td>
</tr>
<tr>
<td>NotchGain</td>
<td>double</td>
<td>Driver notch gain.</td>
</tr>
<tr>
<td>LowpassFrequency</td>
<td>double</td>
<td>Driver low-pass frequency (Hz).</td>
</tr>
</tbody>
</table>

### Output parameters

None.

### Return

(In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.236  

**PositionerDriverPositionOffsetsGet**

**Name**

*PositionerDriverPositionOffsetsGet* – Gets the current value of piezo driver stage and gage position offsets.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
PositionerDriverStatusGet

Name
PositionerDriverStatusGet – Gets the positioner driver status code.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 8.7: “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 “Input Tests Common to all XPS 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).
7.2.1.238  **PositionerDriverStatusStringGet**

**Name**

*PositionerDriverStatusStringGet* – Gets the positioner driver status description.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

This function returns a driver status description from a positioner driver status code.

---

**NOTE**

See section 8.7: “Positioner Driver Status List”.

**Prototype**

```c
#include <xpslib.h>

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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.239  

**PositionerEncoderAmplitudeValuesGet**

**Name**

PositionerEncoderAmplitudeValuesGet – Gets the encoder amplitude values.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner: (-8)
- Checks the encoder type (must be “AnalogInterpolated”): (-8)

**Description**

This function returns the maximum and current amplitudes values (in volts) of the analog encoder input.

---

**NOTE**

The encoder type must be “AnalogInterpolated” in the stages.ini file (“EncoderType” parameter).

---

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

PositionerEncoderCalibrationParametersGet – Gets the encoder calibration parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner (must be not a secondary positioner): (-8)
- Checks 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

**NOTE**

The encoder type must be “AnalogInterpolated” in the stages.ini file (“EncoderType” parameter).

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name

PositionersEncoderIndexDifferenceGet – Gets the distance between the two index encoders (gantry).

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a SingleAxis or an XY): (-8)
- Checks the positioner type (must not be a secondary positioner): (-8)
- Checks the positioner name: (-18)
- Checks the positioner type (must be “gantry”): (-24)
- Checks 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 – Gantries (Secondary Positioners)” section in the XPS User’s Manual.

Prototype

```c
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 “Input Tests Common to all XPS 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).
### 7.2.1.242 PositionerErrorGet

**Name**
PositionerErrorGet – Gets the positioner error code and clears it.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids positioner name: (-18)
- Valids secondary positioner: (-18)

**Description**
This function gets the positioner error code and clears it.
The positioner error codes are listed in section 8.5: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
7.2.1.243  **PositionerErrorRead**

**Name**

*PositionerErrorRead* – Gets the positioner error code without clearing it.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids positioner name: (-18)
- Valids secondary positioner: (-18)

**Description**

This function gets the positioner error code without clearing it.

The positioner error codes are listed in section 8.5: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -18: Positioner Name doesn't exist or unknown command.
- -24: Not available in this configuration (check hardware or configuration).
7.2.1.244 PositionerErrorStringGet

Name
PositionerErrorStringGet – Gets the positioner error description.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

Description
This function returns a positioner error description from a positioner error code.

NOTE
See section 8.5: “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 “Input Tests Common to all XPS Functions”)
• 0: No error.
### 7.2.1.245 PositionerExcitationSignalGet

#### Name
**PositionerExcitationSignalGet** – Gets the currently used parameters of the excitation signal feature.

#### Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Valids positioner name: (-18)

#### Description
This function gets the last configured excitation signal parameters.

#### Prototype
```c
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 “Input Tests Common to all XPS Functions”) 
- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
7.2.1.246  **PositionerExcitationSignalSet**

**Name**

*PositionerExcitationSignalSet* – Sets and activate the signal of excitation.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Is secondary positioner or has a secondary positioner: (-8)
- Checks frequency (must ≥0.1 and ≤0.5/CorrectorISRPeriod): (-17)
- Checks excitation time (must ≥4 * CorrectorISRPeriod): (-17)
- Checks signal amplitude [-Acceleration (Velocity or Voltage) limit to Acceleration (Velocity or Voltage) limit]: (-17)
- Checks type of signal (0, 1, 2 or 3): (-17)
- Valids positioner name: (-18)
- Checks group status (must be READY): (-22)
- Valids 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 (≥0.1 and ≤0.5/CorrectorISRPeriod), Amplitude (>0 and ≤Limit), Time (≥4 * CorrectorISRPeriod)
- Echelon signal mode: Amplitude (>0 and ≤Limit, or <0 and ≥Limit), Time(≥4 * CorrectorISRPeriod)
  + During Time:  Signal = Amplitude
  + End of Time:  Signal = 0
- Random-amplitude signal mode: Amplitude (>0 and ≤Limit), Time(>0), Frequency (≥0.1 and ≤0.5/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 ≤ Limit), Time (≥4 * CorrectorISRPeriod), Frequency (≥0.1 and ≤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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.247  **PositionerExcitationSignalCorrectorOutSet**

**Name**

*PositionerExcitationSignalCorrectorOutSet* – Sets and activates the signal of excitation that is inserted at corrector output.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Is secondary positioner or has a secondary positioner: (-8)
- Checks frequency (must \( \geq 0.1 \) and \( \leq 0.5/\text{CorrectorISRPeriod} \)): (-17)
- Checks excitation time (must \( \geq 4 \times \text{CorrectorISRPeriod} \)): (-17)
- Checks signal amplitude [-.\text{Acceleration} (Velocity or Voltage) limit to \text{Acceleration} (Velocity or Voltage) limit]: (-17)
- Checks type of signal (0, 1, 2 or 3): (-17)
- Valids positioner name: (-18)
- Checks group status (must be \text{READY})): (-22)
- Valids 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 *PositionerExcitationSignalCorrectorOutSet* 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 *PositionerExcitationSignalSignalCorrectorOutSet* 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 (\( \geq 0 \) and \( \leq \text{Limit} \)), Time (\( \geq 4 \times \text{CorrectorISRPeriod} \))

- Echelon signal mode: Amplitude (\( \geq 0 \) and \( \leq \text{Limit} \), or \( < 0 \) and \( \geq \text{Limit} \)), Time(\( \geq 4 \times \text{CorrectorISRPeriod} \))
  
  + During Time:  Signal = Amplitude
  + End of Time:  Signal = 0

- Random-amplitude signal mode: Amplitude (\( \geq 0 \) 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 (\texttt{CorrectorISRPeriod}, default value 0.125 ms), then is filtered with a second order low-pass filter at the cut-off \texttt{Frequency} value.

- Random-pulse-width binary-amplitude signal mode:
  
  \begin{align*}
  \text{Amplitude} & > 0 \text{ and } \leq \text{Limit}, \\
  \text{Time} & \geq 4 \times \text{CorrectorISRPeriod}, \\
  \text{Frequency} & \geq 0.1 \text{ and } \leq 0.5/\text{CorrectorISRPeriod}.
  \end{align*}

  Signal is a sequence of pulses (\texttt{Signal} = \texttt{Amplitude} or = 0) with pulse randomly varied width (multiple of \texttt{Tbase}).

  \texttt{Frequency} is the controlled system band-width (cut-off frequency), necessary for the PRBS (\textit{Pseudo Random Binary Sequence}) function configuration.

  The function non-effective parameters can accept any value, the value 0 is recommended for simplicity.

\begin{center} \textbf{NOTE} \end{center}

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.

\begin{center} \textbf{NOTE} \end{center}

This function does exactly the same that the function \texttt{PositionerExcitationSignalSet()} does, with only one difference between them. The difference is that, with the function \texttt{PositionerExcitationSignalCorrectorOutSet()}, the generated excitation signal is inserted at the corrector output, that can be useful in some cases, like the measurement of robustness transfer function.

\begin{center} \textbf{Prototype} \end{center}

\begin{verbatim}
int PositionerExcitationSignalCorrectorOutSet(
    int SocketID,
    char * FullPositionerName,
    int SignalType,
    double Frequency,
    double Amplitude,
    double Time)

\end{verbatim}

\begin{center} \textbf{Input parameters} \end{center}

\begin{itemize}
  \item \texttt{SocketID} \hspace{2cm} \texttt{int} \hspace{1cm} Socket identifier gets by the \texttt{TCP_ConnectToServer} function.
  \item \texttt{FullPositionerName} \hspace{1cm} \texttt{char *} \hspace{1cm} Positioner name.
  \item \texttt{SignalType} \hspace{1cm} \texttt{int} \hspace{1cm} Type of signal.
  \item \texttt{Frequency} \hspace{1cm} \texttt{double} \hspace{1cm} Frequency (Hz).
  \item \texttt{Amplitude} \hspace{1cm} \texttt{double} \hspace{1cm} Amplitude (acceleration, velocity or voltage unit).
  \item \texttt{Time} \hspace{1cm} \texttt{double} \hspace{1cm} During time (seconds).
\end{itemize}

\begin{center} \textbf{Output parameters} \end{center}

None.
Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.248  PositionerFeedforwardAccDisable

**Name**

PositionerFeedforwardAccDisable – Disables XY External Feed Forward Acceleration signal.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.249  **PositionerFeedforwardAccEnable**

**Name**

*PositionerFeedforwardAccEnable* – Enables XY External Feed forward Acceleration signal.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Valids positioner name: (-18)
- Not allowed due to configuration disabled: (-121)
- Checks the group is ready: (-135)

**Description**

Enable XY External Feed Forward Acceleration signal.
This function is reserved for an XY group.

**Prototype**

```c
int PositionerFeedforwardAccEnable(
    int SocketID,  
    char * PositionerName
)
```

**Input parameters**

- **SocketAddress**  
  int SocketID  
  Socket identifier gets by the  
  “TCP_ConnectToServer” function.
- **PositionerName**  
  char * PositionerName

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.250  **PositionerFeedforwardAccGet**

**Name**

**PositionerFeedforwardAccGet** – Gets parameters of XY external feed forward acceleration signal.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.251  PositionerFeedforwardAccSet

Name

PositionerFeedforwardAccSet – Sets parameters of XY external feed forward acceleration signal.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Valids 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 “Input Tests Common to all XPS 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.
7.2.1.252  PositionerFeedforwardPositionDisable

Name
PositionerFeedforwardPositionDisable – Disables XY External Feed forward Position signal.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.253  PositionerFeedforwardPositionEnable

Name

PositionerFeedforwardPositionEnable – Enables XY External Feed Forward Position signal.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Checks the positioner name: (-18)
- Not allowed due to configuration disabled: (-121)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.254  **PositionerFeedforwardPositionGet**

**Name**

*PositionerFeedforwardPositionGet* – Gets parameters of XY External Feed forward Position signal.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.255  **PositionerFeedforwardPositionSet**

**Name**

*PositionerFeedforwardPositionSet* – Sets parameters of XY External Feed forward Position signal.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Positioner must not be a “Secondary Positioner” and the group must be a XY group: (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
### PositionerGantryEndReferencingPositionGet

**Name**
PositionerGantryEndReferencingPositionGet – Gets the secondary axis position at the end of group home search.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner type (must be a primary positioner): (-8)
- Checks group type: (-8)
  - SingleAxis.
  - SingleAxisTheta.
  - SingleAxisWithClamping.
  - XY.
  - MultipleAxes.
- Checks positioner name: (-18)
- Checks if positioner has a SecondaryPositioner (must be gantry): (-24)
- Checks 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**
```c
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 “Input Tests Common to all XPS 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).
7.2.1.257  PositionerHardInterpolatorFactorGet

**Name**

*PositionerHardInterpolatorFactorGet* – Gets the interpolation factor for position compare mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner (must be not a secondary positioner): (-8)
- Checks 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:

\[
\text{InterpolationFactor} = \text{round}\left(\frac{\text{EncoderScalePitch}}{\text{HardInterpolatorResolution}}\right)
\]

**NOTE**

The encoder type must be “AnalogInterpolated” in the stages.ini file (“EncoderType” parameter).

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

`PositionerHardInterpolatorFactorSet` – Sets the interpolation factor for position compare mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type (must be not a secondary positioner): (-8)
- Checks the encoder type (must be “AnalogInterpolated”): (-8)
- Checks input parameter value: (-17)
- Checks the positioner name: (-18)
- Checks 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:

\[
\text{PositionCompareResolution} = \frac{\text{EncoderScalePitch}}{\text{InterpolationFactor}}
\]

The “InterpolationFactor” value must be define with one of these values:

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>160</td>
</tr>
<tr>
<td>200</td>
</tr>
</tbody>
</table>

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.

This function applies to XPS-Q hardware. It is kept for XPS-D and XPS-RL backward compatibility but has no effect.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
PositionerHardInterpolatorPositionGet

Name
PositionerHardInterpolatorPositionGet – Gets interpolated position from the encoder hard interpolator.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner (must not be a secondary positioner): (-8)
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.1.260  PositionerHardwareStatusGet

**Name**
PositionerHardwareStatusGet – Gets the positioner hardware status code.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 8.6: “Positioner Hardware Status List”.

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0:  No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
7.2.1.261  **PositionerHardwareStatusStringGet**

**Name**

PositionerHardwareStatusStringGet – Gets the positioner error description.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

This function returns the hardware status description from a positioner hardware status code.

---

**NOTE**

See section 8.6: “Positioner Hardware Status List”.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.262  PositionerJogMaximumVelocityAndAccelerationGet

**Name**

PositionerJogMaximumVelocityAndAccelerationGet – Gets the jog maximum velocity and jog maximum acceleration from profiler generator.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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/second²

**Prototype**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
7.2.1.263  **PositionerMagneticTrackPositionAtHomeGet**

**Name**

*PositionerMagneticTrackPositionAtHomeGet* – Gets magnetic track position at home in units.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

**PositionerMaximumVelocityAndAccelerationGet** – Gets the maximum velocity and acceleration from the profiler generators.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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
MaximalAcceleration = ; unit/second²

**Prototype**

```c
def 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/seconds²).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

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

Name
PositionerMotionDoneGet – Gets the motion done parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner (must not be a secondary positioner): (-8)
- Checks 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 “Input Tests Common to all XPS
Functions”)
• 0:  No error.
• -8:  Wrong object type for this command.
7.2.1.266 PositionerMotionDoneSet

**Name**

PositionerMotionDoneSet – Sets the motion done parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
7.2.1.267  

**PositionerMotorDualSinForceBalanceGet**

**Name**


**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if the motor driver interface type is `AnalogDualSinAcceleration` or `AnalogDualSinAccelerationLMI` (error -205 otherwise).

**Description**

This function returns the force balance parameters of an `AnalogDualSinAcceleration` or `AnalogDualSinAccelerationLMI` motor driver interface.

**Prototype**

```c
int PositionerMotorDualSinForceBalanceGet(
    int SocketID,
    char * FullPositionerName,
    double * FirstMotorForceBalance,
    double * SecondMotorForceBalance
)
```

**Input parameters**

- **SocketID** int  Socket identifier gets by the “TCP_ConnectToServer” function.
- **FullPositionerName** char *  Full positioner name *(for example XY.X)*.

**Output parameters**

- **FirstMotorForceBalance**  double *  First force balance ratio.
- **SecondMotorForceBalance**  double *  Second force balance ratio.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0:  No error.
- -205:  Not enable in your configuration.
7.2.1.268  **PositionerMotorDualSinForceBalanceSet**

**Name**


**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if the motor driver interface type is `AnalogDualSinAcceleration` or `AnalogDualSinAccelerationLMI` (error -205 otherwise).

**Description**

This function sets the force balance parameters of an `AnalogDualSinAcceleration` or `AnalogDualSinAccelerationLMI` motor driver interface.

**Prototype**

```c
int PositionerMotorDualSinForceBalanceSet(
    int SocketID,
    char * FullPositionerName,
    double FirstMotorForceBalance,
    double SecondMotorForceBalance
)
```

**Input parameters**

- `SocketID` int Socket identifier gets by the “TCP_ConnectToServer” function.
- `FullPositionerName` char * Full positioner name (for example XY.X).
- `FirstMotorForceBalance` double First force balance ratio (value between 0 and 1).
- `SecondMotorForceBalance` double Second force balance ratio (between 0 and 1).

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -205: Not enable in your configuration.
7.2.1.269  PositionerPositionCompareAquadBAlwaysEnable

Name
PositionerPositionCompareAquadBAlwaysEnable – Enables the AquadB signal in the always mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.270  **PositionerPositionCompareAquadBPrescalerGet**

**Name**

*PositionerPositionCompareAquadBPrescalerGet* – Gets PCO AquadB interpolation factor.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks encoder type (must be “AnalogInterpolated” encoder): (-8)
- Checks parameter value: (-17)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.271 **PositionerPositionCompareAquadBPrescalerSet**

**Name**

`PositionerPositionCompareAquadBPrescalerSet` – Sets PCO AquadB interpolation factor.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks encoder type (must be “AnalogInterpolated” encoder): (-8)
- Checks PCO interpolation factor value: (-17)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.272   PositionerPositionCompareAquadBWindowedGet

Name

PositionerPositionCompareAquadBWindowedGet – Gets the windowed AquadB mode parameters and state.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks the position compare parameters (must be configured): (-23)
- Checks the configured mode type (must be WindowedAquadB): (-24)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.273  **PositionerPositionCompareAquadBWindowedSet**

**Name**

*PositionerPositionCompareAquadBWindowedSet* – Sets the windowed AquadB signal parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks input parameter values: (-17)
  - MinimumPosition < MaximumPosition.
  - MinimumPosition ≥ MinimumTargetPosition.
  - MaximumPosition ≤ MaximumTargetPosition.
- Checks position compare state (must be disabled): (-22)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks 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**

```c
int PositionerPositionCompareAquadBWindowedSet(
    int SocketID,
    char * FullPositionerName,
    double MinimumPosition,
    double MaximumPosition
)
```

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>FullPositionerName</td>
<td>char *</td>
<td>Positioner name.</td>
</tr>
<tr>
<td>MinimumPosition</td>
<td>double</td>
<td>Minimum position (units).</td>
</tr>
<tr>
<td>MaximumPosition</td>
<td>double</td>
<td>Maximum position (units).</td>
</tr>
</tbody>
</table>

**Output parameters**

None.
**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.274  **PositionerPositionCompareDisable**

**Name**

*PositionerPositionCompareDisable* – Disables the position compare mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner (must not be a secondary positioner): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

**PositionerPositionCompareEnable** – Enables the position compare mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner (must not be a secondary positioner): (-8)
- Checks the encoder (“AquadB” or “AnalogInterpolated”): (-8)
- Checks the positioner name: (-18)
- Checks the group state (must be READY): (-22)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**
PositionerPositionCompareGet – Gets the position compare parameters.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner (must not be a secondary positioner): (-8)
- Checks the position compare parameters (must be configured): (-23)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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).
7.2.1.277  

**PositionerPositionComparePulseParametersGet**

**Name**

*PositionerPositionComparePulseParametersGet* – Gets the position compare PCO pulse parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 (μs).

- **EncoderSettlingTime**  
  double *  
  Encoder signal settling time (μs).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- **0**: No error.
- **-8**: Wrong object type for this command.
### 7.2.1.278 PositionerPositionComparePulseParametersSet

**Name**

PositionerPositionComparePulseParametersSet – Sets the position compare PCO pulse parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type (must not be a secondary positioner): (-8)
- Checks input parameter values: (-17)
  - PCOPulseWidth value must equal to 0.2 (default), 1, 2.5 or 10 (µs)
  - EncoderSettlingTime value must equal to 0.075 (default), 1, 4 or 12 (µs)
- Checks position compare state (must be disabled): (-22)
- Checks the position encoder (“AquadB” or “AnalogInterpolated”): (-24)
- Checks 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 loose the PCO position and generate the wrong number of pulses at wrong positions.

For example, if you set the EncoderSettlingTime to 4 µs, 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 ≤62.5e3 * 0.004 / 200 = 1.25 mm/s, otherwise the PCO will not work properly.

How to determine PCO encoder frequency:

1. For AquadB encoder:
   
   \[ PCO \text{ encoder frequency} = \frac{\text{Velocity}}{\text{EncoderResolution}} \]

2. For analog interpolated encoder:
   
   \[ PCO \text{ encoder frequency} = \frac{\text{Velocity} \times \text{HardInterpolatorFactor}}{\text{EncoderScalePitch}} \]

**Example:** XML310 stage (EncoderScalePitch = 0.004 mm, HardInterpolatorFactor = 200). If Velocity = 10 mm/s = >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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.279  

**PositionerPositionCompareScanAccelerationLimitGet**

**Name**

*PositionerPositionCompareScanAccelerationLimitGet* – Gets the position compare scan acceleration limit.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.280  **PositionerPositionCompareScanAccelerationLimitSet**

**Name**

`PositionerPositionCompareScanAccelerationLimitSet` – Sets the position compare acceleration limit.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter values: (-17)
  - PositionCompareScanAccelerationLimit value must >0 and \( \leq \text{MaximumAcceleration (value in stages.ini)} \)
- Checks 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 the (-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**

```c
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/s²).

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0:  No error.
- -17:  Parameter out of range or incorrect.
- -24:  Not available in this configuration (check hardware or configuration).
7.2.1.281  

**PositionerPositionCompareSet**

**Name**

*PositionerPositionCompareSet* – Sets the position compare parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type: (-8)
- Checks input parameter values: (-17)
  - MinimumPosition < MaximumPosition.
  - MinimumPosition ≥ MinimumTargetPosition.
  - MaximumPosition ≤ MaximumTargetPosition.
  - 0 ≤ PositionStep ≤ (MaximumPosition – MinimumPosition)
- Checks position compare state (must be disabled): (-22)
- Checks 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

```c
int PositionerPositionCompareSet(
    int SocketID,
    char * FullPositionerName,
    double MinimumPosition,
    double MaximumPosition,
    double PositionStep,
    bool EnableState
)
```

### Input parameters

- **SocketID**
  - Type: `int`
  - Description: Socket identifier gets by the “TCP_ConnectToServer” function.

- **FullPositionerName**
  - Type: `char *`
  - Description: Positioner name.

- **MinimumPosition**
  - Type: `double`
  - Description: Minimum position (units).

- **MaximumPosition**
  - Type: `double`
  - Description: Maximum position (units).

- **PositionStep**
  - Type: `double`
  - Description: Position step (units).

- **EnableState**
  - Type: `bool`
  - Description: Position compare state (true = enabled or false = disabled).

### Output parameters

None.

### Return

(In addition to the results of “Input Tests Common to all XPS 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).
7.2.1.282 PositionerPreCorrectorExcitationSignalGet

**Name**

PositionerPreCorrectorExcitationSignalGet – Gets the currently used parameters of the pre-corrector excitation signal feature.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Valids positioner name: (-18)

**Description**

This function gets the last configured pre-corrector excitation signal parameters.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- **0**: No error.
- **-8**: Wrong object type for this command.
- **-18**: Positioner Name doesn't exist or unknown command.
7.2.1.283  PositionerPreCorrectorExcitationSignalSet

Name

PositionerPreCorrectorExcitationSignalSet – Configures and activate the signal of pre-corrector excitation.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Is secondary positioner or has a secondary positioner: (-8)
- Checks frequency (must ≥0.1 and ≤0.5/CorrectorISRPeriod): (-17)
- Checks amplitude (must >0): (-17)
- Checks excitation time (must ≥4 * CorrectorISRPeriod): (-17)
- Valids positioner name: (-18)
- Checks group status (must be READY): (-22)
- Valids control loop type: (-24)
- Checks maximum velocity (Amplitude * ω <MaximumVelocity): (-68)
- Checks maximum acceleration (Amplitude * ω² <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 (eg. 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:

\[ \omega = 2\pi F \] (F: excitation frequency)

\[ \text{ExcitationPosition} = A \times \cos(\omega t) - A \] (A: excitation amplitude, t: current time)

\[ \text{ExcitationVelocity} = (-A\omega) \times \sin(\omega t) \]

\[ \text{ExcitationAcceleration} = (-A\omega^2) \times \cos(\omega t) \]

\[ \text{ExcitationJerk} = (A\omega^3) \times \sin(\omega t) \]

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: $\text{Frequency} \geq 0.1 \text{ and } \leq 0.5/\text{CorrectorISRPeriod}$, $\text{Amplitude} > 0$, $\text{Time} \geq 4 \times \text{CorrectorISRPeriod}$.

**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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.284  PositionerRawEncoderPositionGet

**Name**
PositionerRawEncoderPositionGet – Gets the raw encoder position for a positioner.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Valids positioner name: (-18)

**Description**
This function returns the raw encoder position from a corrected position for a positioner.

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
### 7.2.1.285 PositionerSGammaExactVelocityAdjustedDisplacementGet

**Name**

PositionerSGammaExactVelocityAdjustedDisplacementGet – Gets the adjusted displacement to get exact velocity.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the profiler type (must be “SGamma”): (-8)
- Checks 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**

```c
int PositionerSGammaExactVelocityAdjustedDisplacementGet(
    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 * Adjusted displacement (units).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

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

Name

PositionerSGammaParametersGet – Gets the current motion values from the SGamma profiler.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the profiler type (must be “SGamma”): (-8)
- Checks 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/seconds²).
- MinimumJerkTime  double *  Minimum jerk time (seconds).
- MaximumJerkTime  double *  Maximum jerk time (seconds).

Return

(In addition to the results of “Input Tests Common to all XPS Functions”)

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

**Name**

*PositionerSGammaParametersSet* – Sets new motion values for the SGamma profiler.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the profiler type (must be “SGamma”): (-8)
- Checks the positioner type (must not be a secondary positioner): (-8)
- Checks input parameter values: (-17)
  - \[ 0 < NewVelocity \leq MaximumVelocity. \]
  - \[ 0 < NewAcceleration \leq MaximumAcceleration. \]
  - \[ 2 \times ISRProfileGeneratorPeriod \leq NewMinimumJerkTime \text{ and } \leq NewMaximumJerkTime. \]

**Description**

This function defines the new SGamma profiler values that will be used in future displacements.

**Prototype**

```c
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/seconds²).
- **MinimumJerkTime** double  
  Minimum jerk time (seconds).
- **MaximumJerkTime** double  
  Maximum jerk time (seconds).

**Output parameters**

None.
Return (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -17: Parameter out of range or incorrect.
7.2.1.288  PositionerSGammaPreviousMotionTimesGet

Name
PositionerSGammaPreviousMotionTimesGet – Gets the motion and the settling time.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the profiler type (must be “SGamma”): (-8)
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.1.289 **PositionerSGammaVelocityAndAccelerationSet**

**Name**

*PositionerSGammaVelocityAndAccelerationSet* – Sets the SGamma profiler velocity and acceleration.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter values: (-17)
  - \(0 < \text{New Velocity} \leq \text{Maximum Velocity}\)
  - \(0 < \text{New Acceleration} \leq \text{Maximum Acceleration}\)

**Description**

This function sets the SGamma profiler velocity and acceleration that will be used for the future displacements.

**Prototype**

```c
int PositionerSGammaVelocityAndAccelerationSet(int SocketID, double Velocity, double Accelation)
```

**Input parameters**

- **SocketID**: int Socket identifier gets by the “TCP_ConnectToServer” function.
- **Velocity**: double motion velocity (units / seconds).
- **Acceleration**: double motion acceleration (units / seconds²).

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- \(0\): No error.
- \(-17\): Parameter out of range or incorrect.
7.2.1.290  PositionerStageParameterGet

**Name**

PositionerStageParameterGet – Gets a stage parameter value from the stages.ini file.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -13: Wrong parameter type in the command string: char * expected.
- -24: Not available in this configuration (check hardware or configuration).
### 7.2.1.291 PositionerStageParameterSet

**Name**

PositionerStageParameterSet – Saves a stage parameter value into the stages.ini file.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name and the parameter name: (-24)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.292 PositionerTimeFlasherDisable

Name
PositionerTimeFlasherDisable – Disables the time flasher mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.1.293  **PositionerTimeFlasherEnable**

**Name**

`PositionerTimeFlasherEnable` – Enables the time flasher mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type (must not be a secondary positioner): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

#### Name

**PositionerTimeFlasherGet** – Gets the time flasher parameters.

#### Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type (must not be a secondary positioner): (-8)
- Checks the time flasher parameters (must be configured): (-23)
- Checks 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

```c
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 “Input Tests Common to all XPS 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).
7.2.1.295  **PositionerTimeFlasherSet**

**Name**

**PositionerTimeFlasherSet** – Sets the time flasher parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type (must not be a secondary positioner): (-8)
- Checks input parameter values: (-17)
  - MinimumPosition < MaximumPosition.
  - MinimumPosition ≥ MinimumTravelLimit.
  - MaximumPosition ≤ MaximumTravelLimit.
  - 0.0000004 ≤ TimePeriod ≤ 50.0 (Max 2.5 MHz and Min 0.02 Hz)
- Checks the time flasher state (must be disabled): (-22)
- Checks the position encoder (must be used): (-24)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.296  PositionerUserTravelLimitsGet

Name
PositionerUserTravelLimitsGet – Gets the user travel limits.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -118: Not allowed action driver not initialized.
7.2.1.297  **PositionerUserTravelLimitsSet**

**Name**

*PositionerUserTravelLimitsSet* – Sets the user travel limits.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner type (must not be a secondary positioner): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.298 PositionerWarningFollowingErrorGet

Name
PositionerWarningFollowingErrorGet – Gets the warning following error for a positioner.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -18: Positioner Name doesn't exist or unknown command.
7.2.1.299 **PositionerWarningFollowingErrorSet**

**Name**

*PositionerWarningFollowingErrorSet* – Sets value of the warning following error for a positioner.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Checks input parameter values: (-17)
  - \( 0 < \text{WarningFollowingError} \leq \text{FatalFollowingError} \).
- Valids positioner name: (-18)

**Description**

This function sets a new value of the warning following error for a positioner.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.300 Reboot

**Name**
Reboot – Reboots the controller.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
This function reboots the controller.

**NOTE**
This function is not a cold reboot (power off/on), it is a warm (soft) reboot. If an FTP client is connected, this function is not allowed and (-22) error is returned.

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -22: Not allowed action.
7.2.1.301  **RestartApplication**

**Name**

*RestartApplication* – Restarts the controller’s application and avoids hardware reboot.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

This function allows restarting controller applications without hardware reboot.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -22: Not allowed action.
7.2.1.302 SingleAxisSlaveModeDisable

**Name**
SingleAxisSlaveModeDisable – Disables the slave-master mode.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a SingleAxis group): (-8)
- Checks the positioner name: (-18)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.303 SingleAxisSlaveModeEnable

**Name**
SingleAxisSlaveModeEnable – Enables the slave-master mode.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a SingleAxis group): (-8)
- Checks the positioner name: (-18)
- Checks the group name: (-19)
- Group state must be "READY": (-22)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.304  **SingleAxisSlaveParametersGet**

**Name**

*SingleAxisSlaveParametersGet* – Gets the slave parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a SingleAxis group): (-8)
- Checks the positioner name: (-18)
- Checks the slave parameters (must be configured): (-22)

**Description**

This function returns the slave parameters: the master positioner name and the master-slave ratio.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.305  **SingleAxisSlaveParametersSet**

**Name**

*SingleAxisSlaveParametersSet* – Sets the slave parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the master group type: (-8)
- Checks the ratio value (Ratio >0): (-17)
- Checks the master positioner name: (-18)
- Checks the slave parameters (must be configured): (-22)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.306  **SingleAxisThetaClampDisable**

**Name**
SingleAxisThetaClampDisable – unclamps a SingleAxisTheta group.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group): (-8)
- Valids positioner name: (-18)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.307  **SingleAxisThetaClampEnable**

**Name**

*SingleAxisThetaClampEnable* – clamps a SingleAxisTheta group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group): (-8)
- Valids positioner name: (-18)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.308  SingleAxisThetaFeedforwardJerkParametersGet

**Name**

SingleAxisThetaFeedforwardJerkParametersGet – Gets currently used “XY to Theta Feedforward Jerk” feature parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
### 7.2.1.309 SingleAxisThetaFeedforwardJerkParametersSet

**Name**


**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.310  SingleAxisThetaFeedforwardParametersGet

**Name**

*SingleAxisThetaFeedforwardParametersGet* – Gets currently used “XY to Theta Feedforward” feature parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
SingleAxisThetaFeedforwardParametersSet – Sets “XY to Theta Feedforward” feature parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 slows 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.
7.2.1.312 SingleAxisThetaSlaveModeDisable

Name
SingleAxisThetaSlaveModeDisable – Disables the slave-master mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a SingleAxisTheta group): (-8)
- Checks the positioner name: (-18)
- Checks 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 is not the case then the (-22) error 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 “Input Tests Common to all XPS 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.
7.2.1.313 **SingleAxisThetaSlaveModeEnable**

**Name**

*SingleAxisThetaSlaveModeEnable* – Enables the slave-master mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a SingleAxisTheta group): (-8)
- Checks the positioner name: (-18)
- Checks the group name: (-19)
- Group state must be "READY": (-22)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.314  **SingleAxisThetaSlaveParametersGet**

**Name**

*SingleAxisThetaSlaveParametersGet* – Gets the slave parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a SingleAxisTheta group): (-8)
- Checks the positioner name: (-18)
- Checks the slave parameters (must be configured): (-22)

**Description**

This function returns the slave parameters: the master positioner name and the master-slave ratio.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.315 SingleAxisThetaSlaveParametersSet

Name
SingleAxisThetaSlaveParametersSet – Sets the slave parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the master group type: (-8)
- Checks the ratio value (Ratio >0): (-17)
- Checks the master positioner name: (-18)
- Checks the slave parameters (must be configured): (-22)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.316  SpindleSlaveModeDisable

Name
SpindleSlaveModeDisable – Disables the slave-master mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a Spindle group): (-8)
- Checks the positioner name: (-18)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.317  **SpindleSlaveModeEnable**

**Name**

SpindleSlaveModeEnable – Enables the slave-master mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a Spindle group): (-8)
- Checks the positioner name: (-18)
- Checks the group name: (-19)
- Group state must be "READY": (-22)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.318  **SpindleSlaveParametersGet**

**Name**

*SpindleSlaveParametersGet* – Gets the spindle slave parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a Spindle group): (-8)
- Checks the positioner name: (-18)
- Checks the slave parameters (must be configured): (-22)

**Description**

This function returns the slave parameters: the master positioner name and the master-slave ratio.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.319  **SpindleSlaveParametersSet**

**Name**

*SpindleSlaveParametersSet* – Sets the spindle slave parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the master group type: (-8)
- Checks the ratio value (Ratio >0): (-17)
- Checks the master positioner name: (-18)
- Checks the slave parameters (must be configured): (-22)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.320  TCLScriptExecute

Name
TCLScriptExecute – Executes a TCL script.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks TCL file name: (-36)
- Checks TCL interpreter (task loading): (-37)
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -36: Unknown TCL file.
- -37: TCL interpreter does not run.
- -47: Wrong TCL task name: each TCL task name must be different.
7.2.1.321  TCLScriptExecuteAndWait

**Name**
TCLScriptExecuteAndWait – Executes a TCL script and waits until the end of execution.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks TCL file name: (-36)
- Checks TCL interpreter (task loading): (-37)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.322  TCLScriptExecuteWithPriority

**Name**
TCLScriptExecuteWithPriority – Executes a TCL script with TCL task given priority.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Priority mnemonic incorrect: (-17)
- Checks TCL file name: (-36)
- Checks TCL interpreter (task loading): (-37)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.323  TCLScriptKill

**Name**
TCLScriptKill – Kills a TCL script.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks semaphore to use the TCL interpreter: (-37)
- Checks 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**
```c
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 “Input Tests Common to all XPS 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.
7.2.1.324  **TCLScriptKillAll**

**Name**
TCLScriptKillAll – Kills all running TCL scripts.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks TCL interpreter (task loading) and task name: (-38)

**Description**
This function kills all running TCL scripts.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -38: TCL script can not be killed: Wrong task name or task does not run.
### 7.2.1.325 TCLScriptRunningListGet

**Name**
TCLScriptRunningListGet – Gets the list of all TCL processes in progress.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
This function returns the task name’s list of all TCL scripts in progression.

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.326  **TCP_CloseSocket**

**Name**
TCP_CloseSocket – Closes a socket.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
None.
7.2.1.327  TCP_ConnectToServer

**Name**  
TCP_ConnectToServer – Sets TCP/IP communication and opens a socket.

**Input tests**  
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
- SocketID int Socket identifier used in each function.
7.2.1.328  TCP_GetError

**Name**

TCP_GetError – Gets the last error about socket.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

None.
7.2.1.329  **TCP_SetTimeout**

**Name**
TCP_SetTimeout – Sets the timeout for TCP/IP communication.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 Timeout
)
```

**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 “Input Tests Common to all XPS Functions”)
None.
7.2.1.330  TimerGet

**Name**

TimerGet – Gets the number of frequency ticks for the selected timer.

**Input tests**

- Refer to section 7.1: “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 = \(0.125 \text{ ms} = 8 \text{ kHz}\)

\(N\) frequency ticks represents \(N\) corrector periods = \(N \times 0.125 \text{ ms} = N \times 8 \text{ kHz}\)

**NOTE**

“FrequencyTicks” = 0 means that the timer is disabled.

**Prototype**

```c
int TimerGet(
    int SocketID,  // Socket identifier gets by the “TCP_ConnectToServer” function.
    char * TimerName,  // Name of timer.
    int * FrequencyTicks   // Number of frequency ticks.
)
```

**Input parameters**

- **SocketID** int
- **TimerName** char *

**Output parameters**

- **Frequencyticks** int *

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.1.331  TimerSet

**Name**
TimerSet – Sets the number of frequency ticks for the selected timer.

**Input tests**
- Refer to section 7.1: “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 defined the frequency of the timer:

One frequency tick represents a corrector period = \(0.125\) ms = \(8\) kHz

\(N\) frequency ticks represents \(N\) corrector periods = \(N \times 0.125\) ms = \(N \times 8\) kHz

---

**NOTE**

“FrequencyTicks” = 0 means that the timer is disabled.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.1.332  TZEncoderCouplingMatrixGet

Name
TZEncoderCouplingMatrixGet – Gets TZ encoder coupling matrix parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group name: (-19)

Description
This function gets the parameters of encoder coupling matrix for the TZ group.

TZ encoder coupling matrix:

\[
\begin{bmatrix}
\text{Param1} & \text{Param2} & \text{Param3} \\
\text{Param4} & \text{Param5} & \text{Param6} \\
\text{Param7} & \text{Param8} & \text{Param9}
\end{bmatrix}
\]

This API is reserved to TZ group only.

Prototype

```c
int TZEncoderCouplingMatrixGet(
    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 * Encoder coupling matrix coefficient.
- **Param2** double * Encoder coupling matrix coefficient.
- **Param3** double * Encoder coupling matrix coefficient.
- **Param4** double * Encoder coupling matrix coefficient.
- **Param5** double * Encoder coupling matrix coefficient.
- **Param6** double * Encoder coupling matrix coefficient.
- **Param7** double * Encoder coupling matrix coefficient.
- **Param8** double * Encoder coupling matrix coefficient.
- **Param9** double * Encoder coupling matrix coefficient.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -19: GroupName doesn't exist or unknown command.
7.2.1.333 **TZEncoderCouplingMatrixSet**

**Name**

TZEncoderCouplingMatrixSet – Sets TZ encoder coupling matrix parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group name: (-19)

**Description**

This function configures the parameters of encoder coupling matrix for the TZ group.

The values are set in the decoupling matrix in the following order:

\[
\begin{bmatrix}
  \text{Param1} & \text{Param2} & \text{Param3} \\
  \text{Param4} & \text{Param5} & \text{Param6} \\
  \text{Param7} & \text{Param8} & \text{Param9}
\end{bmatrix}
\]

This API is reserved to TZ group only.

**Prototype**

```c
int TZEncoderCouplingMatrixSet(
    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 encoder coupling matrix coefficient.
- **Param2**: double encoder coupling matrix coefficient.
- **Param3**: double encoder coupling matrix coefficient.
- **Param4**: double encoder coupling matrix coefficient.
- **Param5**: double encoder coupling matrix coefficient.
- **Param6**: double encoder coupling matrix coefficient.
- **Param7**: double encoder coupling matrix coefficient.
- **Param8**: double encoder coupling matrix coefficient.
- **Param9**: double encoder coupling matrix coefficient.
Output parameters
None.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.
7.2.1.334  TZEncoderCouplingModeGet

**Name**
TZEncoderCouplingModeGet – Gets TZ encoder coupling mode.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group name: (-19)

**Description**
This function gets the current state of encoder coupling mode (1=Enabled, 0=Disabled). This API is reserved to TZ group only.

**Prototype**

```c
int TZEncoderCouplingModeGet(
    int SocketID,
    char GroupName[250],
    int * Mode
)
```

**Input parameters**

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

**Output parameters**

- **Mode** int *  Encoder coupling mode.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0:  No error.
- -8:  Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.
7.2.1.335  **TZEncoderCouplingModeSet**

**Name**
TZEncoderCouplingModeSet – Sets TZ encoder coupling mode.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group name: (-19)

**Description**
This function configures the mode of encoder coupling.
The values for Mode to set are: 1 (Enabled) or 0 (Disabled).
This API is reserved to TZ group only.

**Prototype**
```c
int TZEncoderCouplingModeSet(
    int SocketID,
    char GroupName[250],
    int Mode
)
```

**Input parameters**
- **SocketID** int  Socket identifier gets by the “TCP_ConnectToServer” function.
- **GroupName** char *  Group name.
- **Mode** int  Encoder coupling mode.

**Output parameters**
None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.
7.2.1.336  **TZFocusModeDisable**

**Name**

TZFocusModeDisable – Disables the TZ group from out of the FOCUS state.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a TZ group): (-8)
- Checks the positioner name: (-18)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.337 **TZFocusModeEnable**

**Name**
TZFocusModeEnable – Enables the TZ group to go in FOCUS state.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a TZ group): (-8)
- Checks the positioner name: (-18)
- Checks 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**
```c
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 “Input Tests Common to all XPS 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.
7.2.1.338  TZMappingModeGet

Name
TZMappingModeGet – Gets TZ mapping mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a TZ group): (-8)
- Checks the group name: (-19)

Description
This function gets the current state of TZ mapping mode (1=Enabled, 0=Disabled).
This API is reserved to TZ group only.

Prototype
int TZMappingModeGet(
    int SocketID,
    char * GroupName,
    int * Mode
)

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

Output parameters
Mode int * TZ mapping mode state.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
• 0: No error.
• -8: Wrong object type for this command.
• -19: GroupName doesn't exist or unknown command.
7.2.1.339  **TZMappingModeSet**

**Name**
TZMappingModeSet – Sets TZ mapping mode.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a TZ group): (-8)
- Checks the group name: (-19)
- Checks if TZMapping feature is enabled in system.ini: (-121)
- Group state must be ”NOTINIT”: (-140)

**Description**
This function configures the mode of TZ mapping.
The values for Mode to set: 1 (Enabled) or 0 (Disabled).
This API is reserved to TZ group only.

**Prototype**

```c
int TZMappingModeSet(
    int SocketID,
    char * GroupName,
    int Mode
)
```

**Input parameters**
- **SocketID** int  Socket identifier gets by the “TCP_ConnectToServer” function.
- **GroupName** char *  TZ group name.
- **Mode** int  TZ mapping mode.

**Output parameters**
None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -19: GroupName doesn't exist or unknown command.
- -121: Function is not allowed due to configuration disabled.
- -140: Function is only allowed in NOTINIT state.
7.2.1.340  TZMotorDecouplingMatrixGet

Name
TZMotorDecouplingMatrixGet – Gets TZ motor decoupling matrix parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group name: (-19)

Description
This function gets the parameters of motor decoupling matrix for the TZ group.
TZ motor decoupling matrix:

\[
\begin{bmatrix}
  \text{Param1} & \text{Param2} & \text{Param3} \\
  \text{Param4} & \text{Param5} & \text{Param6} \\
  \text{Param7} & \text{Param8} & \text{Param9}
\end{bmatrix}
\]

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 “Input Tests Common to all XPS Functions”)

- **0**: No error.
- **-19**: GroupName doesn't exist or unknown command.
7.2.1.341  **TZMotorDecouplingMatrixSet**

**Name**

TZMotorDecouplingMatrixSet – Sets TZ motor decoupling matrix parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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:

\[
\begin{bmatrix}
\text{Param1} & \text{Param2} & \text{Param3} \\
\text{Param4} & \text{Param5} & \text{Param6} \\
\text{Param7} & \text{Param8} & \text{Param9}
\end{bmatrix}
\]

This API is reserved to TZ group only.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
TZMotorDecouplingModeGet – Gets TZ motor decoupling mode.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group name: (-19)

Description
This function gets the current state of motor decoupling mode (1=Enabled, 0=Disabled). This API is reserved to TZ group only.

Prototype
int TZMotorDecouplingModeGet(
    int SocketID,
    char GroupName[250],
    int * Mode
)

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

Output parameters
Mode int * Motor decoupling mode.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
• 0: No error.
• -8: Wrong object type for this command.
• -19: GroupName doesn't exist or unknown command.
7.2.1.343  **TZMotorDecouplingModeSet**

**Name**
TZMotorDecouplingModeSet – Sets TZ motor decoupling mode.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group name: (-19)

**Description**
This function configures the mode of motor decoupling.
The values for Mode to set are: 1 (Enabled) or 0 (Disabled).
This API is reserved to TZ group only.

**Prototype**
```c
int TZMotorDecouplingModeSet(
    int SocketID,
    char GroupName[250],
    int Mode
)
```

**Input parameters**
- **SocketID** int  Socket identifier gets by the “TCP_ConnectToServer” function.
- **GroupName** char *  Group name.
- **Mode** int  Motor decoupling mode.

**Output parameters**
None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0:    No error.
- -8:   Wrong object type for this command.
- -19:  GroupName doesn't exist or unknown command.
7.2.1.344  TZPTExecution

**Name**

TZPTExecution – Executes a PT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length: (-3)
- Checks group type (must be a TZ group): (-8)
- Checks input value (number of executions must >0): (-17)
- Checks group name: (-19)
- Group state must be "READY": (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Checks backlash (must not be enabled): (-46)
- Checks BaseVelocity (stages.ini, must = 0): (-48)
- Checks trajectory file existence or file reading: (-61)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.345  **TZPTLoadToMemory**

**Name**

TZPTLoadToMemory – Loads some lines of PT trajectory to the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory data (data length must $> 0$ and $\leq 400$): (-3) or (-17)
- Checks group type (must be a TZ group): (-8)
- Checks 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, 1).

---

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.346  TZPTParametersGet

Name
TZPTParametersGet – Gets PT trajectory parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a TZ group): (-8)
- Checks group name: (-19)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.347  **TZPTPulseOutputGet**

**Name**

TZPTPulseOutputGet – Gets the configuration of pulse generation of a PT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a TZ group): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.348  **TZPTPulseOutputSet**

**Name**

TZPTPulseOutputSet – Sets the configuration of pulse generation of a PT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a TZ group): (-8)
- Checks group name: (-19)
- Checks 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:

<table>
<thead>
<tr>
<th>GPIO signals</th>
<th>ISA XPS controller</th>
<th>PCI XPS controller (for example XPS-RL) with basic GPIO board</th>
<th>PCI XPS controller (for example XPS-RL) with extended GPIO board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>GPIO2, pin 11</td>
<td>GPIO1.DO6</td>
<td>GPIO5.DO14</td>
</tr>
<tr>
<td>Pulses</td>
<td>GPIO2, pin 12</td>
<td>GPIO1.DO7</td>
<td>GPIO5.DO15</td>
</tr>
</tbody>
</table>

**Example:**

`TZPTPulseOutputSet(GroupName, 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.349  **TZPTResetInMemory**

**Name**

TZPTResetInMemory – Deletes the content of the PT trajectory buffer in the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a TZ group): (-8)
- Checks 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**

```c
int TZPTResetInMemory(
    int SocketID,
    char GroupName[250]
)
```

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>GroupName</td>
<td>char *</td>
<td>Group name.</td>
</tr>
</tbody>
</table>

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
### 7.2.1.350  **TZPTVerification**

**Name**
TZPTVerification – Checks a PT trajectory data file.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length (must ≤250): (-3)
- Checks group type (must be a TZ group): (-8)
- Checks group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Checks BaseVelocity value (must = 0): (-48)
- Checks trajectory file existence and the file format: (-61)
- Checks trajectory (number of elements must >0): (-66)
- Checks velocity (Minimum Velocity ≤Velocity ≤Maximum Velocity): (-68)
- Checks acceleration (Minimum acc. ≤acceleration ≤Maximum acc.): (-69)
- Checks end output velocity (must = 0): (-70)
- Checks 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 algorithm 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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.351  TZPTVerificationResultGet

Name
TZPTVerificationResultGet – Gets the results of the “TZPTVerification” function.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name length (must ≤250): (-3)
- Checks positioner name: (-18)
- Checks 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/s²).
Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.352   **TZPVTExecution**

**Name**

TZPVTExecution – Executes a PVT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length: (-3)
- Checks group type (must be a TZ group): (-8)
- Checks input value (number of executions must >0): (-17)
- Checks group name: (-19)
- Group state must be "READY": (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Checks backlash (must not be enabled): (-46)
- Checks BaseVelocity (stages.ini, must = 0): (-48)
- Checks trajectory file existence or file reading: (-61)
- Checks 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 (-33) error, (-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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.353  **TZPVTLoadToMemory**

**Name**
TZPVTLoadToMemory – Load some lines of PVT trajectory to the controller memory.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Checks group type (must be a TZ group): (-8)
- Checks 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)
TZPVTExecution(socketId,myGroup,FromMemory,1).

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.354  **TZPVTParametersGet**

**Name**
TZPVTParametersGet – Gets PVT trajectory parameters.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a TZ group): (-8)
- Checks group name: (-19)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.355  **TZPVTPulseOutputGet**

**Name**
TZPVTPulseOutputGet – Gets the configuration of pulse generation of a PVT trajectory.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a TZ group): (-8)
- Checks 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 **TZPVTPulseOutputSet()**.
The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

**Example:**
TZPVTPulseOutputSet(MyGroup, 3, 5, 0.01)
TZPVTPulseOutputGet(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**

```c
int TZPVTPulseOutputGet(
    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 “Input Tests Common to all XPS 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.
7.2.1.356  **TZPVTPulseOutputSet**

**Name**

TZPVTPulseOutputSet – Sets the configuration of pulse generation of a PVT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a TZ group): (-8)
- Checks group name: (-19)
- Checks 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:

<table>
<thead>
<tr>
<th>GPIO signals</th>
<th>ISA XPS controller</th>
<th>PCI XPS controller (for example XPS-RL) with basic GPIO board</th>
<th>PCI XPS controller (for example XPS-RL) with extended GPIO board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>GPIO2, pin 11</td>
<td>GPIO1.DO6</td>
<td>GPIO5.DO14</td>
</tr>
<tr>
<td>Pulses</td>
<td>GPIO2, pin 12</td>
<td>GPIO1.DO7</td>
<td>GPIO5.DO15</td>
</tr>
</tbody>
</table>

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual, Appendix / General I/O Description.

**Example:**

TZPVTPulseOutputSet(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**

```c
int TZPVTPulseOutputSet(
    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 “Input Tests Common to all XPS 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.
7.2.1.357  **TZPVTResetInMemory**

**Name**
TZPVTResetInMemory – Deletes the content of the PVT trajectory buffer in the controller memory.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a TZ group): (-8)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.358 **TZPVTVerification**

**Name**
TZPVTVerification – Checks a PVT trajectory data file.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length (must ≤250): (-3)
- Checks group type (must be a TZ group): (-8)
- Checks group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Checks BaseVelocity value (must = 0): (-48)
- Checks trajectory file existence and the file format: (-61)
- Checks trajectory (number of elements must >0): (-66)
- Checks velocity (MinimumVelocity ≤ Velocity ≤ MaximumVelocity): (-68)
- Checks acceleration (MinimumAcc. ≤ acceleration ≤ MaximumAcc.): (-69)
- Checks end output velocity (must = 0): (-70)
- Checks 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 “Input Tests Common to all XPS 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.
### 7.2.1.359 TZPVTVerificationResultGet

**Name**

TZPVTVerificationResultGet – Gets the results of the “TZPVTVerification” function.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name length (must ≤250): (-3)
- Checks positioner name: (-18)
- Checks 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**

```c
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/s²).
**Return** (In addition to the results of “Input Tests Common to all XPS 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.
### 7.2.1.360 TZTrackingCutOffFrequencyGet

**Name**
TZTrackingCutOffFrequencyGet – Gets tracking cut-off frequency of TZ group.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a TZ group): (-8)
- Checks the group (must not be a positioner): (-19)

**Description**
This function gets the tracking cut-off frequency of each positioner of TZ group.

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn't exist or unknown command.
7.2.1.361  **TZTrackingCutOffFrequencySet**

**Name**

TZTrackingCutOffFrequencySet – Sets tracking cut-off frequency of TZ group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a TZ group): (-8)
- Value is out of range: (-17)
- Checks the group (must not be a positioner): (-19)

**Description**

This function sets the tracking cut-off frequency for each positioner of TZ group.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.362 **TZTrackingUserMaximumZZZTargetDifferenceGet**

**Name**

TZTrackingUserMaximumZZZTargetDifferenceGet – Gets tracking maximum ZZZ target difference of TZ group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a TZ group): (-8)
- Checks the group (must not be a positioner): (-19)

**Description**

This function gets the tracking maximum ZZZ target difference of TZ group.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

TZTrackingUserMaximumZZZTargetDifferenceSet – Sets tracking maximum ZZZ target difference for a TZ group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the group type (must be a TZ group): (-8)
- Value is out of range: (-17)
- Checks the group (must not be a positioner): (-19)

**Description**

This function sets the tracking maximum ZZZ target difference for a TZ group.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.364  **XYCrossTalkCompensationMotorDecouplingGet**

**Name**


**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if the cross talk compensation switch is disabled *(firmware.ref): (-121).*

**Description**

This function returns the motor decoupling parameters of XY cross talk compensation.

**Prototype**

```c
int XYCrossTalkCompensationMotorDecouplingGet(
    int SocketID,
    char *GroupName,
    int *CrossTalkCompensationMode,
    double *YToX1FFAccRatio,
    double *YToX2FFAccRatio,
    double *X1ToYFFAccRatio,
    double *X2ToYFFAccRatio
)
```

**Input parameters**

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

**Output parameters**

- **CrossTalkCompensationMode** int * XY cross talk compensation mode (Enabled or Disabled)
- **YToX1FFAccRatio** double * Current Y to X1 feed forward acceleration ratio
- **YToX2FFAccRatio** double * Current Y to X2 feed forward acceleration ratio
- **X1ToYFFAccRatio** double * Current X1 to Y feed forward acceleration ratio
- **X2ToYFFAccRatio** double * Current X2 to Y feed forward acceleration ratio

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -121: Function is not allowed due to configuration disabled
7.2.1.365  \texttt{XYCrossTalkCompensationMotorDecouplingSet}

\textbf{Name}

\texttt{XYCrossTalkCompensationMotorDecouplingSet} – Sets motor decoupling parameters of XY cross talk compensation.

\textbf{Input tests}

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if the cross talk compensation switch is disabled (\texttt{firmware.ref}):( -121).
- Checks if the group is not in NOTINIT nor in DISABLE state: (-215).

\textbf{Description}

This function sets the motor decoupling parameters of XY cross talk compensation.

\textbf{Prototype}

\begin{verbatim}
int XYCrossTalkCompensationMotorDecouplingSet(
    int SocketID,
    char *GroupName,
    int CrossTalkCompensationMode,
    double YToX1FFAccRatio,
    double YToX2FFAccRatio,
    double X1ToYFFAccRatio,
    double X2ToYFFAccRatio
)
\end{verbatim}

\textbf{Input parameters}

- \texttt{SocketID} int Socket identifier gets by the “TCP_ConnectToServer” function.
- \texttt{GroupName} char * Group name.
- \texttt{CrossTalkCompensationMode} int XY cross talk compensation mode (Enabled or Disabled)
- \texttt{YToX1FFAccRatio} double Current Y to X1 feed forward acceleration ratio
- \texttt{YToX2FFAccRatio} double Current Y to X2 feed forward acceleration ratio
- \texttt{X1ToYFFAccRatio} double Current X1 to Y feed forward acceleration ratio
- \texttt{X2ToYFFAccRatio} double Current X2 to Y feed forward acceleration ratio

\textbf{Output parameters}

None.

\textbf{Return} (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -121: Function is not allowed due to configuration disabled
- -215: Not allowed action because group is not in NOTINIT nor in DISABLE state.
7.2.1.366   **XYGroupPositionCorrectedProfilerGet**

**Name**

**XYGroupPositionCorrectedProfilerGet** – Gets the corrected profiler position for all positioners of an XY group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Valids group type (must be an XY group): (-18)
- Valids 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.367  **XYGroupPositionPCORawEncoderGet**

**Name**

XYGroupPositionPCORawEncoderGet – Gets the PCO raw encoder positions for an XY group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Valids group type (must be an XY group): (-18)
- Valids group name: (-19)

**Description**

This function returns the PCO raw encoder positions X and Y from the user specified positions X and Y.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.368  **XYLineArcExecution**

**Name**

*XYLineArcExecution* – Executes a LineArc trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length: (-3)
- Checks group type (must be a XY group): (-8)
- Checks input value (number of executions must >0): (-17)
- Checks input value (Velocity and Acceleration >0): (-17)
- Checks group name: (-19)
- Group state must be "READY": (-22)
- Checks backlash (must not be enabled): (-46)
- Checks BaseVelocity (stages.ini, must = 0): (-48)
- Checks trajectory file existence or file reading: (-61)
- Checks the velocity (Velocity must ≤ TrajectoryMaximumVelocity): (-68)
- Checks the acceleration (Acceleration must ≤ TrajectoryMaximumAcceleration): (-69)
- Checks 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 (-33) error, (-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**

```c
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/s²).
- **ExecutionNumber**: int - Number of trajectory executions.

Output parameters

None.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.369  XYLineArcParametersGet

**Name**

XYLineArcParametersGet – Gets LineArc trajectory parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Checks 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**

```c
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/s²).
- **CurrentElementNumber** int *  
  Currently executing element number.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn’t exist or unknown command.
- -22: Not allowed action.
7.2.1.370  XYLineArcPulseOutputGet

**Name**

XYLineArcPulseOutputGet – Gets the configuration of pulse generation of a LineArc.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

*XYLineArcPulseOutputSet* – Sets the configuration of pulse generation of a LineArc trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Controller initialization failed: (-20)
- Checks 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:

<table>
<thead>
<tr>
<th>GPIO signals</th>
<th>ISA XPS controller</th>
<th>PCI XPS controller (for example XPS-RL) with basic GPIO board</th>
<th>PCI XPS controller (for example XPS-RL) with extended GPIO board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>GPIO2, pin 11</td>
<td>GPIO1.DO6</td>
<td>GPIO5.DO14</td>
</tr>
<tr>
<td>Pulses</td>
<td>GPIO2, pin 12</td>
<td>GPIO1.DO7</td>
<td>GPIO5.DO15</td>
</tr>
</tbody>
</table>

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

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn’t exist or unknown command.
- -22: Not allowed action.
7.2.1.372  XYLineArcVerification

Name

XYLineArcVerification – Checks a LineArc trajectory data file.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length (must ≤250): (-3)
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Checks trajectory file existence and the file format: (-61)
- Checks trajectory element \(\text{Radius} \geq 1e^{-14}\): (-63)
- Checks trajectory element \(\text{SweepAngle} \geq 1e^{-14}\): (-64)
- Checks trajectory element \(|\text{XElementDistance}| \geq 1e^{-14}, |\text{YElementDistance}| \geq 1e^{-14}, \text{TangentOut} \neq 1.797e308\): (-65)
- Checks trajectory (number of elements must >0): (-66)
- Checks 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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.373  XYLineArcVerificationResultGet

**Name**

XYLineArcVerificationResultGet – Gets the results of the “XYLineArcVerification” function.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name length (must ≤250): (-3)
- Checks positioner name: (-18)
- Checks 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**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn’t exist or unknown command.
- -22: Not allowed action.
7.2.1.374  XYMappingGet

**Name**

XYMappingGet – Read data of a line of an XY mapping matrix in the controller.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name (must be a XY positioner): (-18)
- Checks the secondary positioner (must not be a secondary positioner): (-24)
- Checks MappingNumber (must be > 0 and ≤ 1 (standard firmware) or 3 (Precision Platform firmware)): (-9)
- Checks LineNumber (must be ≥ 1 and ≤ MappingLineNumber): (-9)
- Checks the number of parameters (must be > 3 and ≤ MappingColumnNumber+3): (-9)
- Checks 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  0.00234  0.00254  0.00534  0.00192
-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**

```c
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 #LinerNumber of the mapping file #MappingNumber.

Value2  
double *  Data of column #2 of line #LinerNumber of the mapping file #MappingNumber.

…  
double *  One data per column of line #LinerNumber of the mapping file #MappingNumber.

Return (In addition to the results of “Input Tests Common to all XPS 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.
**7.2.1.375 XYMappingSet**

**Name**

XYMappingSet – Change data of a line of an XY mapping matrix in the controller.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the positioner name (must be a XY positioner): (-18)
- Checks the secondary positioner (must not be a secondary positioner): (-24)
- Checks MappingNumber (must be > 0 and ≤ 1 (standard firmware) or 3 (Precision Platform firmware)): (-9)
- Checks LineNumber (must be ≥ 1 and ≤ MappingLineNumber): (-9)
- Checks the number of parameters (must be > 3 and ≤ MappingColumnNumber+3): (-9)
- Checks the group status (must be NOTINIT or DISABLE): (-22)
- Checks mapping enable state (must be Enabled): (-121)
- Checks 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:**

<table>
<thead>
<tr>
<th></th>
<th>0.00</th>
<th>1.00</th>
<th>2.00</th>
<th>3.00</th>
<th>4.00</th>
<th>5.00</th>
<th>6.00</th>
<th>7.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>-1</td>
<td>-0.00192</td>
<td>-0.00534</td>
<td>-0.00254</td>
<td>0.00023</td>
<td>0.00254</td>
<td>0.00534</td>
<td>0.00192</td>
<td>0.00000</td>
</tr>
<tr>
<td>-2</td>
<td>-0.00453</td>
<td>-0.00322</td>
<td>-0.00676</td>
<td>0.00049</td>
<td>0.00676</td>
<td>0.00322</td>
<td>0.00453</td>
<td>0.00000</td>
</tr>
<tr>
<td>-3</td>
<td>-0.00331</td>
<td>-0.00845</td>
<td>-0.00769</td>
<td>0.00102</td>
<td>0.00769</td>
<td>0.00845</td>
<td>0.00331</td>
<td>0.00000</td>
</tr>
<tr>
<td>0</td>
<td>-0.00787</td>
<td>-0.00228</td>
<td>-0.00787</td>
<td>0.00097</td>
<td>0.00787</td>
<td>0.00228</td>
<td>0.00787</td>
<td>0.00000</td>
</tr>
<tr>
<td>1</td>
<td>-0.00232</td>
<td>-0.00210</td>
<td>-0.00342</td>
<td>0.00089</td>
<td>0.00342</td>
<td>0.00210</td>
<td>0.00232</td>
<td>0.00000</td>
</tr>
<tr>
<td>2</td>
<td>-0.00134</td>
<td>-0.00308</td>
<td>-0.00675</td>
<td>0.00101</td>
<td>0.00675</td>
<td>0.00308</td>
<td>0.00134</td>
<td>0.00000</td>
</tr>
<tr>
<td>3</td>
<td>-0.00789</td>
<td>-0.00148</td>
<td>-0.00234</td>
<td>0.00121</td>
<td>0.00234</td>
<td>0.00148</td>
<td>0.00789</td>
<td>0.00000</td>
</tr>
</tbody>
</table>
Prototype

```c
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 #LineNumber of the mapping file #MappingNumber
- **Value2** double Data of column #2 of line #LineNumber of the mapping file #MappingNumber
- **…** double One data per column of line #LineNumber of the mapping file #MappingNumber

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.376  XYPTEExecution

**Name**

XYPTEExecution – Executes a PT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length: (-3)
- Checks group type (must be a XY group): (-8)
- Checks input value (number of executions must >0): (-17)
- Checks group name: (-19)
- Group state must be "READY": (-22)
- Checks backlash (must not be enabled): (-46)
- Checks BaseVelocity (stages.ini, must = 0): (-48)
- Checks trajectory file existence or file reading: (-61)
- Checks 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 “XYPTEVerification” and “XYPTEVerificationResultGet” 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 (-33) error, (-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**

```c
int XYPTEExecution(  
    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 “Input Tests Common to all XPS 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.
7.2.1.377  XYPTLoadToMemory

Name
XYPTLoadToMemory – Load some lines of PT trajectory to the controller memory.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Checks group type (must be a XY group): (-8)
- Checks 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 “Input Tests Common to all XPS 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.
### XYPTParametersGet

**Name**

`XYPTParametersGet` – Gets PT trajectory parameters.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.379  **XYPTPulseOutputGet**

**Name**

XYPTPulseOutputGet – Gets the configuration of pulse generation of a PT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.380  XYPTPulseOutputSet

Name

XYPTPulseOutputSet – Sets the configuration of pulse generation of a PT trajectory.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Checks 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 ouputs:

<table>
<thead>
<tr>
<th>GPIO signals</th>
<th>ISA XPS controller</th>
<th>PCI XPS controller (for example XPS-RL with basic GPIO board)</th>
<th>PCI XPS controller (for example XPS-RL with extended GPIO board)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>GPIO2, pin 11</td>
<td>GPIO1.DO6</td>
<td>GPIO5.DO14</td>
</tr>
<tr>
<td>Pulses</td>
<td>GPIO2, pin 12</td>
<td>GPIO1.DO7</td>
<td>GPIO5.DO15</td>
</tr>
</tbody>
</table>

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual, Appendix / General I/O Description.

Example:

XYPTPulseOutputSet(GroupName, 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 “Input Tests Common to all XPS 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.
7.2.1.381  **XYPTResetInMemory**

**Name**
XYPTResetInMemory – Deletes the content of the PT trajectory buffer in the controller memory.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.382 **XYPTVerification**

**Name**

XYPTVerification – Checks a PT trajectory data file.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length (must ≤250): (-3)
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Checks BaseVelocity value (must = 0): (-48)
- Checks trajectory file existence and the file format: (-61)
- Checks trajectory file existence and the file format: (-66)
- Checks velocity (MinimumVelocity ≤ Velocity ≤ MaximumVelocity): (-68)
- Checks acceleration (MinimumAcc. ≤ Acceleration ≤ MaximumAcc.): (-69)
- Checks end output velocity (must = 0): (-70)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.383  XYPTVerificationResultGet

**Name**

XYPTVerificationResultGet – Gets the results of the “XYPTVerification” function.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name length (must ≤250): (-3)
- Checks positioner name: (-18)
- Checks 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**

```c
int XYPTVerificationResultGet(
    int SocketID,
    char PositionerName[250],
    char * TrajectoryFileName,
    double * MinimumPosition,
    double * MaximumPosition,
    double * MaximumVelocity,
    double * MaximumAcceleration
)
```

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>PositionerName</td>
<td>char *</td>
<td>Positioner name.</td>
</tr>
</tbody>
</table>

**Output parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrajectoryFileName</td>
<td>char *</td>
<td>Examined trajectory file name.</td>
</tr>
<tr>
<td>MinimumPosition</td>
<td>double *</td>
<td>Minimum position (units).</td>
</tr>
<tr>
<td>MaximumPosition</td>
<td>double *</td>
<td>Maximum position (units).</td>
</tr>
<tr>
<td>MaximumVelocity</td>
<td>double *</td>
<td>Maximum velocity (units/s).</td>
</tr>
<tr>
<td>MaximumAcceleration</td>
<td>double *</td>
<td>Maximum acceleration (units/s²).</td>
</tr>
</tbody>
</table>
Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.384  **XYPVTExecution**

**Name**

XYPVTExecution – Executes a PVT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length: (-3)
- Checks group type (must be a XY group): (-8)
- Checks input value (number of executions must >0): (-17)
- Checks group name: (-19)
- Group state must be "READY": (-22)
- Not available in this configuration (PrecisionPlatform firmware only): (-24)
- Checks backlash (must not be enabled): (-46)
- Checks BaseVelocity (stages.ini, must = 0): (-48)
- Checks trajectory file existence or file reading: (-61)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.385 **XYPVTLoadToMemory**

**Name**

XYPVTLoadToMemory – Load some lines of PVT trajectory to the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory data (data length must >0 and ≤400): (-3) or (-17)
- Checks group type (must be a XY group): (-8)
- Checks 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,1).

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.386  XYPVTParametersGet

Name
XYPVTParametersGet – Gets PVT trajectory parameters.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.387  XYPVT PulseOutputGet

**Name**

XYPVT PulseOutputGet – Gets the configuration of pulse generation of a PVT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks 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 XYPVT PulseOutputSet().

The pulse output configuration is defined with a start element, an end element, and a time interval in seconds.

**Example:**

XYPVT PulseOutputSet(MyGroup, 3, 5, 0.01)

XYPVT 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**

```c
int XYPVT 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 “Input Tests Common to all XPS 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.
7.2.1.388  **XYPVT-PulseOutputSet**

**Name**

*XYPVT-PulseOutputSet* – Sets the configuration of pulse generation of a PVT trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Checks 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:

<table>
<thead>
<tr>
<th>GPIO signals</th>
<th>ISA XPS controller</th>
<th>PCI XPS controller (for example XPS-RL) with basic GPIO board</th>
<th>PCI XPS controller (for example XPS-RL) with extended GPIO board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>GPIO2, pin 11</td>
<td>GPIO1.DO6</td>
<td>GPIO5.DO14</td>
</tr>
<tr>
<td>Pules</td>
<td>GPIO2, pin 12</td>
<td>GPIO1.DO7</td>
<td>GPIO5.DO15</td>
</tr>
</tbody>
</table>

To find the GPIO connector pin number from GPIOx.DOy, refer to XPS User’s Manual, Appendix / General I/O Description.

**Example:**

*XYPVT-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**

```c
int XYPVT-PulseOutputSet(
    int SocketID,
    char GroupName[250],
    int StartElement,
    int EndElement,
    double TimeInterval
)
```
### Input parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketID</td>
<td>int</td>
<td>Socket identifier gets by the “TCP_ConnectToServer” function.</td>
</tr>
<tr>
<td>GroupName</td>
<td>char *</td>
<td>Group name.</td>
</tr>
<tr>
<td>StartElement</td>
<td>int</td>
<td>Start pulse element number.</td>
</tr>
<tr>
<td>EndElement</td>
<td>int</td>
<td>End pulse element number.</td>
</tr>
<tr>
<td>TimeInterval</td>
<td>double</td>
<td>Time interval between pulses (seconds).</td>
</tr>
</tbody>
</table>

### Output parameters

None.

### Return

(In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.389  **XYPVTResetInMemory**

**Name**

*XYPVTResetInMemory* – Deletes the content of the PVT trajectory buffer in the controller memory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XY group): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.390 XYPVTVerification

Name
XYPVTVerification – Checks a PVT trajectory data file.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length (must ≤250): (-3)
- Checks group type (must be a XY group): (-8)
- Checks group name: (-19)
- Checks trajectory file existence and the file format: (-61)
- Checks trajectory (number of elements must >0): (-66)
- Checks BaseVelocity value (must = 0): (-48)
- Checks trajectory (number of elements must >0): (-66)
- Checks velocity (Minimum Velocity ≤Velocity ≤Maximum Velocity): (-68)
- Checks acceleration (Minimum acc. ≤acceleration ≤Maximum acc.): (-69)
- Checks end output velocity (must = 0): (-70)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.1.391  **XYPVTVerificationResultGet**

**Name**

**XYPVTVerificationResultGet** – Gets the results of the “XYPVTVerification” function.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name length (must ≤250): (-3)
- Checks positioner name: (-18)
- Checks 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**

```c
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/s²).
Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.392  **XYZGroupPositionCorrectedProfilerGet**

**Name**

**XYZGroupPositionCorrectedProfilerGet** – Gets the corrected profiler position for all positioners of an XYZ group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Valids group type (must be an XYZ group): (-18)
- Valids 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

```c
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 “Input Tests Common to all XPS 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.
7.2.1.393 XYZGroupPositionPCORawEncoderGet

Name
XYZGroupPositionPCORawEncoderGet – Gets the PCO raw encoder positions of an XYZ group.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type: (-8)
- Valids group type (must be a XYZ group): (-18)
- Valids 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 “Input Tests Common to all XPS 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.
7.2.1.394  **XYZSplineExecution**

**Name**
XYZSplineExecution – Executes a Spline trajectory.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length: (-3)
- Checks group type (must be a XYZ group): (-8)
- Checks input value (Velocity and Acceleration must >0): (-17)
- Checks group name: (-19)
- Group state must be "READY": (-22)
- Checks backlash (must not be enabled): (-46)
- Checks BaseVelocity (stages.ini, must = 0): (-48)
- Checks trajectory file existence or file reading: (-61)
- Checks message queue: (-71)
- Checks the velocity (Velocity ≤ TrajectoryMaximumVelocity): (-68)
- Checks 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**

```c
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/s²).

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.1.395  XYZSplineParametersGet

Name

XYZSplineParametersGet – Gets Spline trajectory parameters.

Input tests

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XYZ group): (-8)
- Checks group name: (-19)
- Checks 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

```c
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/s²).
- CurrentElementNumber: int * Currently executing element number.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn’t exist or unknown command.
- -22: Not allowed action.
7.2.1.396  XYZSplinePulseOutputGet

**Name**

*XYZSplinePulseOutputGet* – Gets the configuration of pulse generation of a Spline trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XYZ group): (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

**Name**

**XYZSplinePulseOutputSet** – Sets the configuration of pulse generation of a Spline trajectory.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks group type (must be a XYZ group): (-8)
- Checks group name: (-19)
- Checks 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:

<table>
<thead>
<tr>
<th>GPIO signals</th>
<th>ISA XPS controller</th>
<th>PCI XPS controller (for example XPS-RL with basic GPIO board)</th>
<th>PCI XPS controller (for example XPS-RL with extended GPIO board)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>GPIO2, pin 11</td>
<td>GPIO1.DO6</td>
<td>GPIO5.DO14</td>
</tr>
<tr>
<td>Pulses</td>
<td>GPIO2, pin 12</td>
<td>GPIO1.DO7</td>
<td>GPIO5.DO15</td>
</tr>
</tbody>
</table>

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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -8: Wrong object type for this command.
- -19: Group name doesn’t exist or unknown command.
- -22: Not allowed action.
7.2.1.398  **XYZSplineVerification**

**Name**

**XYZSplineVerification** – Checks a Spline trajectory data file.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks trajectory file name length (must ≤250): (-3)
- Checks group type (must be a XYZ group): (-8)
- Checks group name: (-19)
- Checks trajectory file existence and the file format: (-61)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.1.399  XYZSplineVerificationResultGet

**Name**

*XYZSplineVerificationResultGet* – Gets the results of the “XYZSplineVerification” function.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks positioner name length (must ≤250): (-3)
- Checks positioner name: (-18)
- Checks 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**

```c
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/s²).

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -3: String too long.
- -18: Positioner name doesn’t exist or unknown command.
- -22: Not allowed action.
7.2.2 Extended Functions

The functions described in this section are for a specific controller configuration, please contact Newport for further information.

7.2.2.1 AbortMove

**Name**
AbortMove – abort the motion or the jog in progress for the XY group.

**Input tests**
- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -27: Move Aborted.
7.2.2.2 **EndJog**

**Name**

EndJog – Disables the jog mode in the XY group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.2.3 GetAccParams

Name
GetAccParams – Gets acceleration parameters for X and Y axes.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)
• 0: No error.
7.2.2.4 GetBrakeState

Name
GetBrakeState – Gets the brake status.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)
• 0: No error.
### 7.2.2.5 GetCurrentPosition

**Name**

GetCurrentPosition – Gets all current positions of X, X2 and Y.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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: µm
- X1 and X2 positions: counts
- Y and X laser positions: counts

**Prototype**

```c
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 µm.
- **x_position_um** double * X position in µm.
- **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 “Input Tests Common to all XPS 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.
7.2.2.6 GetGantryMode

**Name**
GetGantryMode – Gets Gantry mode.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the set option: (-17)
- Checks the group name is valid: (-19)
- Verifies 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -19: GroupName doesn't exist or unknown command.
- -22: Not allowed action.
7.2.2.7  GetJogAcceleration

Name
GetJogAcceleration – Gets the acceleration set by “SetJogAcceleration”.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Checks the positioner name: (-18)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.2.8 GetJogVelocity

**Name**
GetJogVelocity – Changes “on the fly” the velocity in the jog mode.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Checks the positioner name: (-18)
- Checks 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**
```c
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 µm/s.
- **Yvelocity** double * user jog velocity for Y in µm/s.
- **joystickAckTimeout_ms** int * user jog velocity acknowledge timeout in ms.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.9 **GetPistonState**

**Name**

GetPistonState – Gets current status of Piston and Lift Pin.

**Input tests**

- Refer to section 7.1: “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 signal meaning depending on Piston…SignalState value:**

<table>
<thead>
<tr>
<th>PistonCommandSignalState</th>
<th>Direct</th>
<th>Inverted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piston Command Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Released</td>
<td>Engaged</td>
</tr>
<tr>
<td>1</td>
<td>Engaged</td>
<td>Released</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PistonEngagedSignalState</th>
<th>Direct</th>
<th>Inverted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piston Engaged Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not Engaged</td>
<td>Engaged</td>
</tr>
<tr>
<td>1</td>
<td>Engaged</td>
<td>Not Engaged</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PistonReleasedSignalState</th>
<th>Direct</th>
<th>Inverted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piston Released Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not Released</td>
<td>Released</td>
</tr>
<tr>
<td>1</td>
<td>Released</td>
<td>Not Released</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lift Pin UP Status</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not activated (Lift Pin is down)</td>
</tr>
<tr>
<td>1</td>
<td>Activated (Lift Pin is up)</td>
</tr>
</tbody>
</table>

**Prototype**

```c
int GetPistonState(
    int SocketID,
    int * CommandState,
    int * isEngaged,
    int * isReleased,
    int * LiftPinUPInterlock
)
```
**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).
LiftPinUPInterlock int * Status of the Lift Pin UP (0 or 1).

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.10  **GetVarX**

**Name**

GetVarX – Sets new value for a specified parameter name from stages.ini for Y positioner.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter value: (-17)
- Checks positioner name: (-18)
- Verifies the positioner is from a XY group: (-22)

**Description**

Get parameter value from stages.ini file for the X positioner.

**Prototype**

```c
int GetVarX(  
    int SocketID,  
    char * ParameterName,  
    double * ParameterValue  
)
```

**Input parameters**

<table>
<thead>
<tr>
<th>SocketID</th>
<th>int</th>
<th>Socket identifier gets by the “TCP_ConnectToServer” function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParameterName</td>
<td>char *</td>
<td>Stages.ini parameter name.</td>
</tr>
</tbody>
</table>

**Output parameters**

| ParameterValue | double * | The value to be set for the “ParameterName”. |

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
7.2.2.11  GetVarXSecondary

Name
GetVarXSecondary – Gets the configured value of the parameter name from stages.ini for X secondary positioner.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter value: (-17)
- Checks positioner name: (-18)
- Verifies 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
7.2.2.12  GetVarY

Name
GetVarY – Gets the configured value of the parameter name from stages.ini for Y positioner.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter value: (-17)
- Checks positioner name: (-18)
- Verifies 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn’t exist or unknown command.
- -22: Not allowed action.
7.2.2.13 GetVelParams

**Name**
GetVelParams – Gets current velocity for X and Y axes.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**
```c
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 µm/s.
Yvelocity double * Y velocity in µm/s.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.2.14 GetVerCommand

Name
GetVerCommand – Return firmware version.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.2.15  **GetXVelParams**

**Name**
GetXVelParams – This API returns the current velocity parameter in µm/s for X axis.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks motion profile: (-8)

**Description**
This API returns the current velocity parameter in µm/s for X axis.

**Prototype**
```c
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 µm/s.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.2.16  **GetYVelParams**

**Name**

GetYVelParams – This API returns the current velocity parameter in µm/s for Y axis.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks motion profile: (-8)

**Description**

This API returns the current velocity parameter in µm/s for Y axis.

**Prototype**

```c
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 µm/s.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)

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

Name
GetZone – Gets current parameters of the defined circle zone.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 μm. 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 μm.
y_center_um double * Y center in μm.
radius_um double * Radius in μm.
hysteresis_um double * Hysteresis in μm.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.18 InitializeAndHomeX

Name
InitializeAndHomeX – Do home search on X axis.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the set option: (-17)
- Verifies this command is allowed: (-22)
- Checks state of physical ends of run: (-113)
- Checks the opened socked is valid: (-200)
- Checks 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 “Input Tests Common to all XPS 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.
- -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.
7.2.2.19  **InitializeAndHomeXY**

**Name**

InitializeAndHomeXY – Do home search on X then Y axis.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the set option: (-17)
- Verifies this command is allowed: (-22)
- Checks state of physical ends of run: (-113)
- Checks the opened socket is valid: (-200)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.2.20  **InitializeAndHomeY**

**Name**

InitializeAndHomeY – Do home search on Y axis.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Verifies this command is allowed: (-22)
- Checks state of physical ends of run: (-113)
- Checks the opened socket is valid: (-200)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
- -113: Both ends of run activated.
- -200: Invalid socket.
- -202: Not allowed action due to an external motion interlock.
- -205: Not enabled in your configuration.
- -208: Not allowed action because piston is engaged.
- -1004: Zygo signal is not present.
### 7.2.2.21 MoveAbsolute

**Name**

**MoveAbsolute** – Moves the stage to the end position. The positions are defined in um.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Valids object type (group or positioner): (-8)
- Verifies target position in relation with the travel limits: (-17)
  - TargetPosition ≥MinimumTargetPosition.
  - TargetPosition ≤MaximumTargetPosition.
- Valids positioner name: (-18)
- Valids 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**

```c
int MoveAbsolute(
    int SocketID,
    double PositionAbsoluteX_um,
    double PositionAbsoluteY_um
)
```
Input parameters

- **SocketID**: int
  - Socket identifier gets by the “TCP_ConnectToServer” function.
- **PositionAbsoluteX_\_um**: double
  - Target position in µm for X axis.
- **PositionAbsoluteY_\_um**: double
  - Target position in µm for Y axis.

Output parameters

- None.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.22  MoveSlice

**Name**

**MoveSlice** – Executes a slice move on an XY group.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Verifies target position in relation with the travel limits: (-17)
- Checks 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).

![Diagram of MoveSlice](image)

**Prototype**

```c
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 µm.
- **Xend_um**  
  *double*  
  x target position in µm.
- **ScanAngle_urad**  
  *double*  
  Scan angle in µrad.
Output parameters
None.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.23  

**RequestType1**

**Name**
RequestType1 – Gets data collection Type 1.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
This function returns the data collection of type1.

**Prototype**
```c
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 µm.
1DPoS charhex32 * 1D+ current position in µm.
XFe charhex32 * X following error in µm.
XPos charhex32 * Yaw following error in µm.
YawFe charhex32 * Yaw current position in µm.
YFe charhex32 * Y following error in µm.
YPos charhex32 * Y current position in µm.
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -100: Internal error (memory allocation error, …).
- -136: Wrong parameter type in the command string: charhex32 * expected.
7.2.2.24   **RequestType2**

**Name**
RequestType2— Gets data collection Type 2.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
This function returns the data collection of type2.

**Prototype**
```c
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 “Input Tests Common to all XPS 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 8.9: “Error List”.

7.2.2.25  RequestType3

Name
RequestType3—Gets data collection Type 3.

Input tests
- Refer to section 7.1: “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 * ZygoSignalStrength1,
    charhex32 * ZygoSignalStrength2
)

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.
ZygoSignalStrength1 charhex32 * Zygo Signal Strength 1.
ZygoSignalStrength2 charhex32 * Zygo Signal Strength 2.

Return (In addition to the results of “Input Tests Common to all XPS 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 8.9: “Error List”.
7.2.2.26 SetAccParams

**Name**
SetAccParams – Sets acceleration parameters for X and Y axes.

**Input tests**
- Refer to section 7.1: “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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.2.27  **SetBrake**

**Name**
SetBrake—Sets Brake command.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
- -205: Not enable in your configuration.
7.2.2.28 **SetGantryMode**

**Name**

*SetGantryMode* – Sets Gantry mode (Option0, Option1 or Option2).

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the set option: (-17)
- Checks the group name is valid: (-19)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.2.29  **SetJogAcceleration**

**Name**

*SetJogAcceleration* – Changes the acceleration parameters in the jog mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Checks the positioner name: (-18)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.2.30 **SetJogVelocity**

**Name**

*SetJogVelocity* – Changes “on the fly” the velocity in the jog mode.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Checks the positioner name: (-18)
- Checks 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**

```c
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 µm/s.
- **Yvelocity** double: user jog velocity for Y in µm/s.
- **joystickAckTimeout_ms** int: user jog velocity acknowledge timeout in ms.

**Output parameters**

None.
Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.31 SetPiston

**Name**
SetPiston – Sets command to activate piston “Engaged” or “Released”.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
- -205: Not enable in your configuration.
7.2.2.32  **SetVarX**

**Name**
SetVarX – Sets new value for a specified parameter name from stages.ini for X positioner.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter value: (-17)
- Checks positioner name: (-18)
- Verifies 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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
7.2.2.33  **SetVarXSecondary**

**Name**

SetVarXSecondary – Sets new value for a specified parameter name from stages.ini for X secondary positioner.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter value: (-17)
- Checks positioner name: (-18)
- Verifies 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**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
7.2.2.34  **SetVarY**

**Name**
SetVarY – Sets new value for a specified parameter name from stages.ini for Y positioner.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter value: (-17)
- Checks positioner name: (-18)
- Verifies 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**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
- -18: Positioner Name doesn't exist or unknown command.
- -22: Not allowed action.
7.2.2.35  **SetVelParams**

**Name**
SetVelParams – Sets acceleration parameters for X and Y axes.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks motion profile: (-8)

**Description**
This API updates the current velocity parameters in µm/s for X and Y axes.

**Prototype**
```c
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 µm/s.
- **Yvelocity** double Y velocity in µm/s.

**Output parameters**
None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.2.36 SetXVelParams

**Name**
SetXVelParams – Sets velocity for X axis.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks motion profile: (-8)

**Description**
This API updates the velocity parameter in µm/s for X axis.

**Prototype**
```c
int SetXVelParams(
    int SocketID,  
    double Xvelocity
)
```

**Input parameters**
- **SocketID** int Socket identifier gets by the “TCP_ConnectToServer” function.
- **Xvelocity** double X velocity in µm/s.

**Output parameters**
None.

**Return** (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.2.37  SetYVelParams

Name
SetYVelParams – Sets velocity for Y axis.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks motion profile: (-8)

Description
This API updates the velocity parameter in µm/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 µm/s.

Output parameters
None.

Return (In addition to the results of “Input Tests Common to all XPS Functions”)
- 0: No error.
- -8: Wrong object type for this command.
7.2.2.38  SetZone

Name
SetZone – Sets parameters to define the circle zone.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 µm.
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 µm.
y_center_um  double  Y center in µm.
radius_um  double  Radius in µm.
hysteresis_um  double  Hysteresis in µm.

Output parameters
None.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.39  **StartJog**

**Name**
StartJog – Enables the jog mode in XY group.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Checks the positioner name: (-18)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.2.40  \textbf{WaitMotionEnd}

\textbf{Name}
\texttt{WaitMotionEnd} – Wait the end of move (XY group only).

\textbf{Input tests}
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Group must be “XY”: (-8)
- Checks expected position after motion: (-211)

\textbf{Description}
This function allows waiting the real end of move (after a \texttt{GroupMoveAbsolute}, \texttt{GroupMoveRelative} or \texttt{GroupMoveSlice}).

\begin{center}
\textbf{NOTE}
This function is available only for an XY group.
\end{center}

\textbf{Prototype}
\begin{verbatim}
int WaitMotionEnd(
    int SocketID,
    char * GroupName,
    double TimeOutMs,
    double YPosition,
    double XPosition
)
\end{verbatim}

\textbf{Input parameters}
\begin{itemize}
\item \texttt{SocketID} int Socket identifier gets by the \texttt{“TCP_ConnectToServer”} function.
\item \texttt{GroupName} char * XY Group name.
\item \texttt{TimeOutMs} double Time out in milliseconds.
\item \texttt{YPosition} double Y position to check in microns.
\item \texttt{XPosition} double X position to check in microns.
\end{itemize}

\textbf{Output parameters}
None.

\textbf{Return} (In addition to the results of “Input Tests Common to all XPS Functions”)
\begin{itemize}
\item 0: No error.
\item -17: Parameter out of range or incorrect.
\item -22: Not allowed action.
\item -211: Not expected position after motion.
\end{itemize}
7.2.2.41  **ZygoADCDiagnosticStatusGet**

**Name**

ZygoADCDiagnosticStatusGet – Gets the diagnostic ADC status.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks Axis number and Zygo axis number: (-17)

**Description**

Returns the diagnostic ADC status in relation to the ADC Mux number.

**Prototype**

```c
int ZygoADCDiagnosticStatusGet(
    int SocketID,
    int Axis,
    int ADCMuxNumber,
    char * ADCDiagStatus
)
```

**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**

- **ADCDiagStatus** char * Raw value from Diag ADC Register.

**Return** (In addition to the results of “Input Tests Common to all XPS 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 8.9: “Error List”.


7.2.2.42  ZygoAmplitudeGet

**Name**
ZygoAmplitudeGet – Gets the reference channel status and measuring channel amplitude.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**
Returns the reference channel status and the measuring channel amplitudes.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
7.2.2.43 **ZygoConnectToServer**

**Name**
ZygoConnectToServer – Connect ZYGO to server.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the object type of this command is valid: (-8)
- Checks Zygo is enabled: (-205)

**Description**
This function opens a TCP/IP communication with a ZMI Measuring board.

**Prototype**

```c
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 “Input Tests Common to all XPS 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.
7.2.2.44  **ZygoDisconnectFromServer**

**Name**

*ZygoDisconnectFromServer* – Disconnect ZYGO from server.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

Close the TCP/IP communication opened with the ZYGO ZMI box.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.2.45  ZygoErrorStatusGet

**Name**

ZygoErrorStatusGet – Gets the ZYGO error status code via Ethernet.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks Zygo axis value: (-17)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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 8.9: “Error List”.

---

**Newport**

645  

EDH0373En1024 — 03/18
7.2.2.46  ZygoErrorStatusStringGet

Name
ZygoErrorStatusStringGet – Gets the ZYGO axis error status description via Ethernet.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
• 0: No error.
• -17: Parameter out of range or incorrect.
7.2.2.47  **ZygoEthernetCommunicationStatusGet**

**Name**

ZygoEthernetCommunicationStatusGet – Gets ZYGO Ethernet Connection Status.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.

**Description**

Get the Ethernet Communication Status: Connected or Not connected.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0: No error.
7.2.2.48  ZygoGetPEGLastCommunicationTime

**Name**

*ZygoGetPEGLastCommunicationTime* – Gets the last communication time to configure Zygo PEG.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks the XPS controller configuration: (-4)

**Description**

This function gets the last communication time to configure Zygo PEG in seconds.

**Prototype**

```c
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 “Input Tests Common to all XPS Functions”)

- 0:  No error.
- -4:  Unknown command.
7.2.2.49  

**ZygoGetVerInterfero**

**Name**

*ZygoGetVerInterfero* – Gets firmware version of ZMI system (ZYGO interferometer) via Ethernet.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS 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.
7.2.2.50  ZygoInterferometerStatusGet

Name
ZygoInterferometerStatusGet – Gets ZYGO interferometer status.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS Functions”)
- 0: No error.
7.2.2.51  ZygoPositionGet

Name
ZygoPositionGet – Gets positions of Y and X axes from ZMI system via Ethernet.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS 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 8.9: “Error List”.

7.2.2.52 **ZygoReadLong**

**Name**

ZygoReadLong – Read a long register from a ZMI Measuring board.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS 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 8.9: “Error List”. 


7.2.2.53  ZygoReadWord

Name
ZygoReadWord – Read a long register from a ZMI Measuring board.

Input tests
- Refer to section 7.1: “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 “Input Tests Common to all XPS 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 8.9: “Error List”.


7.2.2.54  **ZygoRegisterGet**

**Name**

ZygoRegisterGet – Gets register value from P2 ZYGO board.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS Functions”)

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

Name
ZygoRegisterSet – Sets register value from P2 ZYGO board.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
• 0: No error.
• -8: Wrong object type for this command.
7.2.2.56  ZygoReset

**Name**
ZygoReset – Reset all Zygo axes.

**Input tests**
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks if Zygo mode is enabled: (-205)
- Checks Zygo TCP Server Connexion: (-1001)

**Description**
This function sends commands to Zygo ZMI 2402 to reset all Zygo axes (#1 and #2).

**Prototype**
```c
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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -205: Not enable in your configuration.
- -1001: The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
7.2.2.57  

**ZygoResetX**

**Name**

ZygoResetX – Reset X axis via Ethernet.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.2.58  ZygoResetY

**Name**

ZygoResetY – Reset Y axis via Ethernet.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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.

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 3-7  Axis Reset Conditions and Required Time

<table>
<thead>
<tr>
<th>Enable Reset Finds Velocity</th>
<th>RFV Mode Aux Control 0</th>
<th>Reset Time</th>
<th>Max Velocity during reset</th>
<th>Max Acceleration during reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>22 μs + RD</td>
<td>≈0.1 m/s</td>
<td>10 g</td>
</tr>
<tr>
<td>1</td>
<td>0 (Diagnostic)</td>
<td>2.0 ms + RD</td>
<td>≈2.1 m/s</td>
<td>0.1 g</td>
</tr>
<tr>
<td>1</td>
<td>1 (Default)</td>
<td>210 μs + RD</td>
<td>≈2.1 m/s</td>
<td>10 g</td>
</tr>
</tbody>
</table>

---

### Table 3-8  Reset Delay

<table>
<thead>
<tr>
<th>RD2</th>
<th>RD1</th>
<th>RD0</th>
<th>Reset Delay (RD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2 μs</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6 μs</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>26 μs</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>102 μs</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>410 μs</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.5 ms</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6.5 ms</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>26.2 ms</td>
</tr>
</tbody>
</table>
**Prototype**

```c
int ZygoResetY(
    int SocketID
)
```

**Input parameters**

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

**Output parameters**

None.

**Return** (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.59  ZygoSendAndReceive

**Name**

ZygoSendAndReceive – Send a command and read a response from ZYGO box.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS 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 8.9: “Error List”.

7.2.2.60 **ZygoSetOffsetX**

**Name**

ZygoSetOffsetX – Sets offset value for X axis via Ethernet.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter type: (-127)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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.
7.2.2.61  ZygoSetOffsetY

Name
ZygoSetOffsetY – Sets offset value for Y axis via Ethernet.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter type: (-127)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.2.62 **ZygoSetPEGParams**

**Name**

ZygoSetPEGParams – Sets PEG parameters (ZYGO System) via Ethernet.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks input parameter type: (-127)
- Checks 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 “Input Tests Common to all XPS 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.
7.2.2.63  ZygoStartBoardP2

**Name**

ZygoStartBoardP2 – Starts ZYGO P2 board.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)

- **0:** No error.
- **-17:** Parameter out of range or incorrect.
- **-1001:** The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.
7.2.2.64  ZygoStartInterferometer

Name
ZygoStartInterferometer – Start ZYGO interferometer.

Input tests
- Refer to section 7.1: “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 * ADCDiagStatus
)

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
ADCDiagStatus char * Raw value from Diag ADC Register.

Return (In addition to the results of “Input Tests Common to all XPS 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.
7.2.2.65  **ZygoStatusGet**

**Name**

*ZygoStatusGet* – Gets the ZYGO axis status code via Ethernet.

**Input tests**

- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks Zygo axis value: (-17)
- Checks 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**

```c
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 “Input Tests Common to all XPS 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 8.9: “Error List”.

EDH0373En1024 — 03/18  668
7.2.2.66  ZygoStatusStringGet

Name
ZygoStatusStringGet – Gets the ZYGO axis status description via Ethernet.

Input tests
- Refer to section 7.1: “Input Tests Common to all XPS Functions”.
- Checks 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 “Input Tests Common to all XPS Functions”)
- 0: No error.
- -17: Parameter out of range or incorrect.
7.2.2.67  **ZygoWriteLong**

**Name**  
*ZygoWriteLong* – Write data to a long register of ZMI Measuring board.

**Input tests**  
- Refer to section 7.1: “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:*

\[
\text{ZygoWriteLong(AXIS1, A, 5000)}
\]

ZMI Measuring board Response:  
OK

**Prototype**  
\[
\text{int ZygoWriteLong(}
\text{int SocketID, char * AxisNum, char * Register, char * Data, char * Response)}
\]

**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 “Input Tests Common to all XPS 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 8.9: “Error List”.
7.2.2.68  ZygoWriteWord

**Name**

ZygoWriteWord – Write data to a word register of ZMI Measuring board.

**Input tests**

- Refer to section 7.1: “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**

```c
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 “Input Tests Common to all XPS 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 8.9: “Error List”.
## 8.0 Lists and Tables for XPS Functions

### 8.1 Event Triggers List

<table>
<thead>
<tr>
<th>[Actor.]</th>
<th>[Category.]</th>
<th>Event Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Immediate</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Always</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Timer</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MotionStart</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MotionEnd</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MotionState</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MotionDone</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MotionDoneTimeout</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConstantVelocityStart</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConstantVelocityEnd</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConstantVelocityState</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConstantAccelerationStart</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConstantAccelerationEnd</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConstantDecelerationStart</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConstantDecelerationEnd</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConstantDecelerationState</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TrajectoryStart</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TrajectoryEnd</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TrajectoryState</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ElementNumberStart</td>
<td>Element index 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ElementNumberState</td>
<td>Element index 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TrajectoryPulse</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TrajectoryPulseState</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DILowState</td>
<td>Bit index 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIHighState</td>
<td>Bit index 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DILowHigh</td>
<td>Bit index 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIHighLow</td>
<td>Bit index 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIToggled</td>
<td>Bit index 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADCHighLimit</td>
<td>Value 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADCLowLimit</td>
<td>Value 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADCInWindow</td>
<td>min max 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADCOutWindow</td>
<td>min max 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PositionerError</td>
<td>Mask 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PositionerHardwareStatus</td>
<td>Mask 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ExcitationSignalStart</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ExcitationSignalEnd</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WarningFollowingError</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WaitForPositionLeftToRight</td>
<td>Target position 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WaitForPositionRightToLeft</td>
<td>Target position 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WaitForPositionToggled</td>
<td>Target position 0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DoubleGlobalArrayEqual</td>
<td>Global variable number value 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DoubleGlobalArrayDifferent</td>
<td>Global variable number value 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DoubleGlobalArrayInferiorOrEqual</td>
<td>Global variable number value 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DoubleGlobalArraySuperiorOrEqual</td>
<td>Global variable number value 0 0</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Value</td>
<td>Min</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>DoubleGlobalArrayInferior</td>
<td>Global variable</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DoubleGlobalArraySuperior</td>
<td>Global variable</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DoubleGlobalArrayInWindow</td>
<td>Global variable</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>DoubleGlobalArrayOutWindow</td>
<td>Global variable</td>
<td>min</td>
<td>max</td>
</tr>
</tbody>
</table>
# 8.2 Actions List

<table>
<thead>
<tr>
<th>Group</th>
<th>Positioner</th>
<th>GPIO</th>
<th>TimerX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DACSet.CurrentPosition</td>
<td>Positioner name</td>
<td>Gain</td>
</tr>
<tr>
<td></td>
<td>DACSet.CurrentVelocity</td>
<td>Positioner name</td>
<td>Gain</td>
</tr>
<tr>
<td></td>
<td>DACSet.SetpointPosition</td>
<td>Positioner name</td>
<td>Gain</td>
</tr>
<tr>
<td></td>
<td>DACSet.SetpointVelocity</td>
<td>Positioner name</td>
<td>Gain</td>
</tr>
<tr>
<td></td>
<td>DACSet.SetpointAcceleration</td>
<td>Positioner name</td>
<td>Gain</td>
</tr>
<tr>
<td></td>
<td>DACSet.Value</td>
<td>Value</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>DOPulse</td>
<td>Mask</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>DOToggle</td>
<td>Mask</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>DOSet</td>
<td>Mask</td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>EventRemove</td>
<td>Trigger identifier</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ExecuteCommand</td>
<td>Function name</td>
<td>Parameters (Between {} and separator is the semi-column)</td>
</tr>
<tr>
<td></td>
<td>ExecuteTCLScript</td>
<td>TCL file name</td>
<td>Task name</td>
</tr>
<tr>
<td></td>
<td>ExternalGatheringRun</td>
<td>Nb of points</td>
<td>Divisor</td>
</tr>
<tr>
<td></td>
<td>GatheringOneData</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>GatheringRun</td>
<td>Nb of points</td>
<td>Divisor</td>
</tr>
<tr>
<td></td>
<td>GatheringRunAppend</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>GatheringStop</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>GlobalArrayDoubleSet</td>
<td>Global variable number</td>
<td>Double value</td>
</tr>
<tr>
<td></td>
<td>GlobalArrayStringSet</td>
<td>Global variable number</td>
<td>String value</td>
</tr>
<tr>
<td></td>
<td>KillTCLScript</td>
<td>Task name</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>MoveAbort</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>MoveAbortFast</td>
<td>Deceleration multiplier</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SynchronizeProfiler</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
8.3 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)`

**XPS-Q 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
**XPS-RL or XPS-D 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
8.4 External Gathering Data Types

PositionerName.ExternalLatchPosition

**XPS-Q Hardware:**
- GPIO2.ADC1
- GPIO2.ADC2
- GPIO2.ADC3
- GPIO2.ADC4
- GPIO2.DAC1
- GPIO2.DAC2
- GPIO2.DAC3
- GPIO2.DAC4

**XPS-RL or XPS-D 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
## 8.5 Positioner Error List

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Mask</th>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>General inhibition detected</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Fatal following error detected</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Home search time out</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>Motion done time out</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>Requested position exceed travel limits in trajectory or slave mode</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>Requested velocity exceed maximum value in trajectory or slave mode</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>Requested acceleration exceed maximum value in trajectory or slave mode</td>
</tr>
<tr>
<td>7</td>
<td>168</td>
<td>Clamping incoherence</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>Minus end of run activated</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>Plus end of run activated</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>Minus end of run glitch</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
<td>Plus end of run glitch</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
<td>Encoder quadrature error</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
<td>Encoder frequency and coherancy error</td>
</tr>
<tr>
<td>14</td>
<td>16384</td>
<td>Sine and Cosine radius error</td>
</tr>
<tr>
<td>15</td>
<td>32768</td>
<td>X or Y correction is out of encoder correction limits</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>Hard interpolator encoder error</td>
</tr>
<tr>
<td>17</td>
<td>131072</td>
<td>Hard interpolator encoder quadrature error</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
<td>First driver in fault</td>
</tr>
<tr>
<td>21</td>
<td>2097152</td>
<td>Second driver in fault</td>
</tr>
<tr>
<td>22</td>
<td>4194304</td>
<td>AqB and Sine/Cosine out of phase</td>
</tr>
<tr>
<td>24</td>
<td>16777216</td>
<td>Home search mechanical zero inconsistency</td>
</tr>
<tr>
<td>25</td>
<td>33554432</td>
<td>Interferometer no signal error on axis or reference</td>
</tr>
<tr>
<td>26</td>
<td>67108864</td>
<td>Interferometer glitch error on axis or reference</td>
</tr>
<tr>
<td>27</td>
<td>134217728</td>
<td>Fatal internal error</td>
</tr>
<tr>
<td>28</td>
<td>268435456</td>
<td>GPIO transfer error</td>
</tr>
<tr>
<td>29</td>
<td>536870912</td>
<td>I2C transfer error</td>
</tr>
</tbody>
</table>

**NOTE**

The most significant bit is always set to 1. So, all positioner errors are negative.
### 8.6 Positioner Hardware Status List

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Mask</th>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>General inhibition detected</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>ZM high level</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>Minus end of run activated</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>Plus end of run activated</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>Minus end of run glitch</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
<td>Plus end of run glitch</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
<td>Encoder quadrature error</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
<td>Encoder frequency or coherancy error</td>
</tr>
<tr>
<td>14</td>
<td>16384</td>
<td>Sine and Cosine radius error</td>
</tr>
<tr>
<td>15</td>
<td>32768</td>
<td>X or Y correction is out of encoder correction limits</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>Hard interpolator encoder error</td>
</tr>
<tr>
<td>17</td>
<td>131072</td>
<td>Hard interpolator encoder quadrature error</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
<td>First driver in fault</td>
</tr>
<tr>
<td>21</td>
<td>2097152</td>
<td>Second driver in fault</td>
</tr>
<tr>
<td>22</td>
<td>4194304</td>
<td>First driver powered on</td>
</tr>
<tr>
<td>23</td>
<td>8388608</td>
<td>Second driver powered on</td>
</tr>
<tr>
<td>24</td>
<td>16777216</td>
<td>Interferometer no signal error on axis or reference</td>
</tr>
<tr>
<td>25</td>
<td>33554432</td>
<td>Interferometer glitch error on axis or reference</td>
</tr>
<tr>
<td>26</td>
<td>67108864</td>
<td>PCO error (underrun)</td>
</tr>
<tr>
<td>27</td>
<td>134217728</td>
<td>PCO pulses ended</td>
</tr>
<tr>
<td>28</td>
<td>268435456</td>
<td>GPIO External gathering error</td>
</tr>
<tr>
<td>29</td>
<td>536870912</td>
<td>I2C transfer error</td>
</tr>
<tr>
<td>30</td>
<td>1073741824</td>
<td>External gathering error</td>
</tr>
</tbody>
</table>

**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 Inhibition 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.
## 8.7 Positioner Driver Status List

<table>
<thead>
<tr>
<th>Bit</th>
<th>code</th>
<th>DRV00x</th>
<th>DRV01</th>
<th>DRV02x</th>
<th>D6U</th>
<th>DRV03</th>
<th>DRVP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>Short circuit</td>
<td>Short circuit</td>
<td>Short circuit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>Broken fuse</td>
<td>Broken fuse</td>
<td>Broken fuse</td>
<td></td>
<td>Voltage out of range</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>Thermistor fault</td>
<td>Thermistor fault</td>
<td></td>
<td></td>
<td></td>
<td>Over temperature</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>Initialization error</td>
<td>Initialization error</td>
<td>Initialization error</td>
<td>Initialization error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>e</td>
<td>I²T</td>
<td>I²T</td>
<td>I²T</td>
<td>Dynamic error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>f</td>
<td>Current limit</td>
<td>Current limit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td>TG is opened</td>
<td>No stage connected</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>h</td>
<td>Inhibition input</td>
<td>Inhibition input</td>
<td>Inhibition input</td>
<td>Inhibition input</td>
<td>Inhibition input</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>i</td>
<td>Driver in fault</td>
<td>Driver in fault</td>
<td>Driver in fault</td>
<td>Driver in fault</td>
<td>Driver in fault</td>
<td></td>
</tr>
</tbody>
</table>
## 8.8 Group Status List

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NOTINIT state</td>
</tr>
<tr>
<td>1</td>
<td>NOTINIT state due to an emergency brake: see positioner status</td>
</tr>
<tr>
<td>2</td>
<td>NOTINIT state due to an emergency stop: see positioner status</td>
</tr>
<tr>
<td>3</td>
<td>NOTINIT state due to a following error during homing</td>
</tr>
<tr>
<td>4</td>
<td>NOTINIT state due to a following error</td>
</tr>
<tr>
<td>5</td>
<td>NOTINIT state due to an homing timeout</td>
</tr>
<tr>
<td>6</td>
<td>NOTINIT state due to a motion done timeout during homing</td>
</tr>
<tr>
<td>7</td>
<td>NOTINIT state due to a KillAll command</td>
</tr>
<tr>
<td>8</td>
<td>NOTINIT state due to an end of run after homing</td>
</tr>
<tr>
<td>9</td>
<td>NOTINIT state due to an encoder calibration error</td>
</tr>
<tr>
<td>10</td>
<td>Ready state due to an AbortMove command</td>
</tr>
<tr>
<td>11</td>
<td>Ready state from homing</td>
</tr>
<tr>
<td>12</td>
<td>Ready state from motion</td>
</tr>
<tr>
<td>13</td>
<td>Ready State due to a MotionEnable command</td>
</tr>
<tr>
<td>14</td>
<td>Ready state from slave</td>
</tr>
<tr>
<td>15</td>
<td>Ready state from jogging</td>
</tr>
<tr>
<td>16</td>
<td>Ready state from analog tracking</td>
</tr>
<tr>
<td>17</td>
<td>Ready state from trajectory</td>
</tr>
<tr>
<td>18</td>
<td>Ready state from spinning</td>
</tr>
<tr>
<td>19</td>
<td>Ready state due to a group interlock error during motion</td>
</tr>
<tr>
<td>20</td>
<td>Disable state</td>
</tr>
<tr>
<td>21</td>
<td>Disabled state due to a following error on ready state</td>
</tr>
<tr>
<td>22</td>
<td>Disabled state due to a following error during motion</td>
</tr>
<tr>
<td>23</td>
<td>Disabled state due to a motion done timeout during moving</td>
</tr>
<tr>
<td>24</td>
<td>Disabled state due to a following error on slave state</td>
</tr>
<tr>
<td>25</td>
<td>Disabled state due to a following error on jogging state</td>
</tr>
<tr>
<td>26</td>
<td>Disabled state due to a following error during trajectory</td>
</tr>
<tr>
<td>27</td>
<td>Disabled state due to a motion done timeout during trajectory</td>
</tr>
<tr>
<td>28</td>
<td>Disabled state due to a following error during analog tracking</td>
</tr>
<tr>
<td>29</td>
<td>Disabled state due to a slave error during motion</td>
</tr>
<tr>
<td>30</td>
<td>Disabled state due to a slave error on slave state</td>
</tr>
<tr>
<td>31</td>
<td>Disabled state due to a slave error on jogging state</td>
</tr>
<tr>
<td>32</td>
<td>Disabled state due to a slave error during trajectory</td>
</tr>
<tr>
<td>33</td>
<td>Disabled state due to a slave error during analog tracking</td>
</tr>
<tr>
<td>34</td>
<td>Disabled state due to a slave error on ready state</td>
</tr>
<tr>
<td>35</td>
<td>Disabled state due to a following error on spinning state</td>
</tr>
<tr>
<td>36</td>
<td>Disabled state due to a slave error on spinning state</td>
</tr>
<tr>
<td>37</td>
<td>Disabled state due to a following error on auto-tuning</td>
</tr>
<tr>
<td>38</td>
<td>Disabled state due to a slave error on auto-tuning</td>
</tr>
<tr>
<td>39</td>
<td>Disable state due to an emergency stop on auto-tuning state</td>
</tr>
<tr>
<td>40</td>
<td>Emergency braking</td>
</tr>
<tr>
<td>41</td>
<td>Motor initialization state</td>
</tr>
<tr>
<td>42</td>
<td>Not referenced state</td>
</tr>
<tr>
<td>43</td>
<td>Homing state</td>
</tr>
<tr>
<td>44</td>
<td>Moving state</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>45</td>
<td>Trajectory state</td>
</tr>
<tr>
<td>46</td>
<td>Slave state due to a SlaveEnable command</td>
</tr>
<tr>
<td>47</td>
<td>Jogging state due to a JogEnable command</td>
</tr>
<tr>
<td>48</td>
<td>Analog tracking state due to a TrackingEnable command</td>
</tr>
<tr>
<td>49</td>
<td>Analog interpolated encoder calibrating state</td>
</tr>
<tr>
<td>50</td>
<td>NOTINIT state due to a mechanical zero inconsistency during homing</td>
</tr>
<tr>
<td>51</td>
<td>Spinning state due to a SpinParametersSet command</td>
</tr>
<tr>
<td>52</td>
<td>NOTINIT state due to a clamping timeout</td>
</tr>
<tr>
<td>53</td>
<td>Clamped</td>
</tr>
<tr>
<td>54</td>
<td>Ready state from clamped</td>
</tr>
<tr>
<td>55</td>
<td>Disabled state due to a following error during clamped</td>
</tr>
<tr>
<td>56</td>
<td>Disabled state due to a motion done timeout during clamped</td>
</tr>
<tr>
<td>57</td>
<td>NOTINIT state due to a group interlock error on not reference state</td>
</tr>
<tr>
<td>58</td>
<td>NOTINIT state due to a group interlock error during homing</td>
</tr>
<tr>
<td>59</td>
<td>NOTINIT state due to a master/slave error during homing</td>
</tr>
<tr>
<td>60</td>
<td>Auto-tuning state</td>
</tr>
<tr>
<td>61</td>
<td>Scaling calibration state</td>
</tr>
<tr>
<td>62</td>
<td>Ready state from auto-tuning</td>
</tr>
<tr>
<td>63</td>
<td>NOTINIT state from scaling calibration</td>
</tr>
<tr>
<td>64</td>
<td>NOTINIT state due to a scaling calibration error</td>
</tr>
<tr>
<td>65</td>
<td>Excitation signal generation state</td>
</tr>
<tr>
<td>66</td>
<td>Disable state due to a following error on excitation signal generation state</td>
</tr>
<tr>
<td>67</td>
<td>Disable state due to a master/slave error on excitation signal generation state</td>
</tr>
<tr>
<td>68</td>
<td>Disable state due to an emergency stop on excitation signal generation state</td>
</tr>
<tr>
<td>69</td>
<td>Ready state from excitation signal generation</td>
</tr>
<tr>
<td>70</td>
<td>Focus state</td>
</tr>
<tr>
<td>71</td>
<td>Ready state from focus</td>
</tr>
<tr>
<td>72</td>
<td>Disable state due to a following error on focus state</td>
</tr>
<tr>
<td>73</td>
<td>Disable state due to a master/slave error on focus state</td>
</tr>
<tr>
<td>74</td>
<td>Disable state due to an emergency stop on focus state</td>
</tr>
<tr>
<td>75</td>
<td>NOTINIT state due to a group interlock error</td>
</tr>
<tr>
<td>76</td>
<td>Disable state due to a group interlock error during moving</td>
</tr>
<tr>
<td>77</td>
<td>Disable state due to a group interlock error during jogging</td>
</tr>
<tr>
<td>78</td>
<td>Disable state due to a group interlock error on slave state</td>
</tr>
<tr>
<td>79</td>
<td>Disable state due to a group interlock error during trajectory</td>
</tr>
<tr>
<td>80</td>
<td>Disable state due to a group interlock error during analog tracking</td>
</tr>
<tr>
<td>81</td>
<td>Disable state due to a group interlock error during spinning</td>
</tr>
<tr>
<td>82</td>
<td>Disable state due to a group interlock error on ready state</td>
</tr>
<tr>
<td>83</td>
<td>Disable state due to a group interlock error on auto-tuning state</td>
</tr>
<tr>
<td>84</td>
<td>Disable state due to a group interlock error on excitation signal generation state</td>
</tr>
<tr>
<td>85</td>
<td>Disable state due to a group interlock error on focus state</td>
</tr>
<tr>
<td>86</td>
<td>Disabled state due to a motion done timeout during jogging</td>
</tr>
<tr>
<td>87</td>
<td>Disabled state due to a motion done timeout during spinning</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>96</td>
<td>Disabled state due to a motion done timeout during slave mode</td>
</tr>
<tr>
<td>97</td>
<td>Disabled state due to a ZYGO error during motion</td>
</tr>
<tr>
<td>98</td>
<td>Disabled state due to a master/slave error during trajectory</td>
</tr>
<tr>
<td>99</td>
<td>Disable state due to a ZYGO error on jogging state</td>
</tr>
<tr>
<td>100</td>
<td>Disabled state due to a ZYGO error during analog tracking</td>
</tr>
<tr>
<td>101</td>
<td>Disable state due to a ZYGO error on auto-tuning state</td>
</tr>
<tr>
<td>102</td>
<td>Disable state due to a ZYGO error on excitation signal generation state</td>
</tr>
<tr>
<td>103</td>
<td>Disabled state due to a ZYGO error on ready state</td>
</tr>
<tr>
<td>104</td>
<td>Driver initialization</td>
</tr>
<tr>
<td>105</td>
<td>Jitter initialization</td>
</tr>
<tr>
<td>106</td>
<td>Not initialized state due to an error with GroupKill or KillAll command</td>
</tr>
</tbody>
</table>
## Error List

<table>
<thead>
<tr>
<th>code</th>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Error to ignore</td>
</tr>
<tr>
<td>1</td>
<td>TCL interpreter error: wrong syntax</td>
</tr>
<tr>
<td>0</td>
<td>Successful command</td>
</tr>
<tr>
<td>-1</td>
<td>Busy socket: previous command not yet finished</td>
</tr>
<tr>
<td>-2</td>
<td>TCP timeout</td>
</tr>
<tr>
<td>-3</td>
<td>String command too long</td>
</tr>
<tr>
<td>-4</td>
<td>Unknown command</td>
</tr>
<tr>
<td>-5</td>
<td>Not allowed due to a positioner error</td>
</tr>
<tr>
<td>-7</td>
<td>Wrong format in the command string</td>
</tr>
<tr>
<td>-8</td>
<td>Wrong object type for this command</td>
</tr>
<tr>
<td>-9</td>
<td>Wrong number of parameters in the command</td>
</tr>
<tr>
<td>-10</td>
<td>Wrong parameter type in the command string</td>
</tr>
<tr>
<td>-11</td>
<td>Wrong parameters type in the command string: word or word * expected</td>
</tr>
<tr>
<td>-12</td>
<td>Wrong parameter type in the command string: bool or bool * expected</td>
</tr>
<tr>
<td>-13</td>
<td>Wrong parameter type in the command string: char * expected</td>
</tr>
<tr>
<td>-14</td>
<td>Wrong parameter type in the command string: double or double * expected</td>
</tr>
<tr>
<td>-15</td>
<td>Wrong parameter type in the command string: int or int * expected</td>
</tr>
<tr>
<td>-16</td>
<td>Wrong parameter type in the command string: unsigned int or unsigned int * expected</td>
</tr>
<tr>
<td>-17</td>
<td>Parameter out of range</td>
</tr>
<tr>
<td>-18</td>
<td>Positioner Name doesn't exist</td>
</tr>
<tr>
<td>-19</td>
<td>Group Name doesn't exist or unknown command</td>
</tr>
<tr>
<td>-20</td>
<td>Fatal Error during initialization, read the error.log file for more details</td>
</tr>
<tr>
<td>-21</td>
<td>Controller in initialization</td>
</tr>
<tr>
<td>-22</td>
<td>Not allowed action</td>
</tr>
<tr>
<td>-23</td>
<td>Position compare not set</td>
</tr>
<tr>
<td>-24</td>
<td>Not available in this configuration</td>
</tr>
<tr>
<td>-25</td>
<td>Following Error</td>
</tr>
<tr>
<td>-26</td>
<td>Emergency signal</td>
</tr>
<tr>
<td>-27</td>
<td>Move Aborted</td>
</tr>
<tr>
<td>-28</td>
<td>Home search timeout</td>
</tr>
<tr>
<td>-29</td>
<td>Mnemonic gathering type doesn't exist</td>
</tr>
<tr>
<td>-30</td>
<td>Gathering not started</td>
</tr>
<tr>
<td>-31</td>
<td>Home position is out of user travel limits</td>
</tr>
<tr>
<td>-32</td>
<td>Gathering not configurated</td>
</tr>
<tr>
<td>-33</td>
<td>Motion done timeout</td>
</tr>
<tr>
<td>-35</td>
<td>Not allowed: home preset outside travel limits</td>
</tr>
<tr>
<td>-36</td>
<td>Unknown TCL file</td>
</tr>
<tr>
<td>-37</td>
<td>TCL interpreter doesn't run</td>
</tr>
<tr>
<td>-38</td>
<td>TCL script can't be killed</td>
</tr>
<tr>
<td>-39</td>
<td>Mnemonic action doesn't exist</td>
</tr>
<tr>
<td>-40</td>
<td>Mnemonic event doesn't exist</td>
</tr>
<tr>
<td>-41</td>
<td>Slave-Master mode not configurated</td>
</tr>
<tr>
<td>-42</td>
<td>Jog value out of range</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>-43</td>
<td>Gathering running</td>
</tr>
<tr>
<td>-44</td>
<td>Slave error disabling master</td>
</tr>
<tr>
<td>-45</td>
<td>End of run activated</td>
</tr>
<tr>
<td>-46</td>
<td>Not allowed action due to backlash</td>
</tr>
<tr>
<td>-47</td>
<td>Wrong TCL task name: each TCL task name must be different</td>
</tr>
<tr>
<td>-48</td>
<td>BaseVelocity must be null</td>
</tr>
<tr>
<td>-49</td>
<td>Inconsistent mechanical zero during home search</td>
</tr>
<tr>
<td>-50</td>
<td>Motor initialization error: check InitializationAcceleration</td>
</tr>
<tr>
<td>-51</td>
<td>Spin value out of range</td>
</tr>
<tr>
<td>-52</td>
<td>Group interlock</td>
</tr>
<tr>
<td>-53</td>
<td>Not allowed action due to a group interlock</td>
</tr>
<tr>
<td>-54</td>
<td>Error during file writing or file doesn't exist</td>
</tr>
<tr>
<td>-55</td>
<td>Error during file reading or file doesn't exist</td>
</tr>
<tr>
<td>-56</td>
<td>Wrong trajectory element type</td>
</tr>
<tr>
<td>-57</td>
<td>Wrong XY trajectory element arc radius</td>
</tr>
<tr>
<td>-58</td>
<td>Wrong XY trajectory element sweep angle</td>
</tr>
<tr>
<td>-59</td>
<td>Trajectory line element discontinuity error or new element is too small</td>
</tr>
<tr>
<td>-60</td>
<td>Trajectory doesn't contain any element or not loaded</td>
</tr>
<tr>
<td>-61</td>
<td>Velocity on trajectory is too high</td>
</tr>
<tr>
<td>-62</td>
<td>Acceleration on trajectory is too high</td>
</tr>
<tr>
<td>-63</td>
<td>Final velocity on trajectory is not zero</td>
</tr>
<tr>
<td>-64</td>
<td>Error write or read from message queue</td>
</tr>
<tr>
<td>-65</td>
<td>Error during trajectory initialization</td>
</tr>
<tr>
<td>-66</td>
<td>End of file</td>
</tr>
<tr>
<td>-67</td>
<td>Error file parameter key not found</td>
</tr>
<tr>
<td>-68</td>
<td>Time delta of trajectory element is negative or null</td>
</tr>
<tr>
<td>-69</td>
<td>Event not configured</td>
</tr>
<tr>
<td>-70</td>
<td>Action not configured</td>
</tr>
<tr>
<td>-71</td>
<td>Event buffer is full</td>
</tr>
<tr>
<td>-72</td>
<td>Event ID not defined</td>
</tr>
<tr>
<td>-73</td>
<td>Secondary positioner index is too far from first positioner</td>
</tr>
<tr>
<td>-74</td>
<td>Focus socket not reserved or closed</td>
</tr>
<tr>
<td>-75</td>
<td>Focus event scheduler is busy</td>
</tr>
<tr>
<td>-76</td>
<td>Error of executing an optional module</td>
</tr>
<tr>
<td>-77</td>
<td>Error of stopping an optional module</td>
</tr>
<tr>
<td>-78</td>
<td>Error of unloading an optional module</td>
</tr>
<tr>
<td>-79</td>
<td>Fatal external module load: see error.log</td>
</tr>
<tr>
<td>-80</td>
<td>Internal error (memory allocation error, …)</td>
</tr>
<tr>
<td>-81</td>
<td>Relay Feedback Test failed: No oscillation</td>
</tr>
<tr>
<td>-82</td>
<td>Relay Feedback Test failed: Signal too noisy</td>
</tr>
<tr>
<td>-83</td>
<td>Error of tuning process initialization</td>
</tr>
<tr>
<td>-84</td>
<td>Error of scaling calibration initialization</td>
</tr>
<tr>
<td>-85</td>
<td>Wrong user name or password</td>
</tr>
<tr>
<td>-86</td>
<td>This function requires to be logged in with Administrator rights</td>
</tr>
<tr>
<td>-87</td>
<td>The TCP/IP connection was closed by an administrator</td>
</tr>
<tr>
<td>-88</td>
<td>Group need to be homed at least once to use this function (distance measured during home search)</td>
</tr>
<tr>
<td>Code</td>
<td>Error Message</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td>-110</td>
<td>Execution not allowed for Gantry configuration</td>
</tr>
<tr>
<td>-111</td>
<td>Gathering buffer is full</td>
</tr>
<tr>
<td>-112</td>
<td>Error of excitation signal generation initialization</td>
</tr>
<tr>
<td>-113</td>
<td>Both ends of run activated</td>
</tr>
<tr>
<td>-114</td>
<td>Clamping timeout</td>
</tr>
<tr>
<td>-115</td>
<td>Function is not supported by current hardware</td>
</tr>
<tr>
<td>-116</td>
<td>Error during external driver initialization, read error.log file for more details</td>
</tr>
<tr>
<td>-117</td>
<td>Function is only allowed in DISABLED group state</td>
</tr>
<tr>
<td>-118</td>
<td>Not allowed action driver not initialized</td>
</tr>
<tr>
<td>-119</td>
<td>Position is outside of travel limits on secondary positioner</td>
</tr>
<tr>
<td>-120</td>
<td>Warning following error during move with position compare enabled</td>
</tr>
<tr>
<td>-121</td>
<td>Function is not allowed due to configuration disabled</td>
</tr>
<tr>
<td>-122</td>
<td>Data incorrect (wrong value, wrong format, wrong order or inexistent)</td>
</tr>
<tr>
<td>-123</td>
<td>Action not allowed, an Administrator is already logged in</td>
</tr>
<tr>
<td>-124</td>
<td>Error during move of secondary positioner: check positioners errors for details</td>
</tr>
<tr>
<td>-125</td>
<td>Check tcl task name is not empty</td>
</tr>
<tr>
<td>-126</td>
<td>Wrong parameter type in the command string: short or short * expected</td>
</tr>
<tr>
<td>-127</td>
<td>Wrong parameter type in the command string: long or long * expected</td>
</tr>
<tr>
<td>-128</td>
<td>Wrong parameter type in the command string: unsigned short or unsigned short * expected</td>
</tr>
<tr>
<td>-129</td>
<td>Wrong parameter type in the command string: unsigned long or unsigned long * expected</td>
</tr>
<tr>
<td>-130</td>
<td>Wrong parameter type in the command string: float or float * expected</td>
</tr>
<tr>
<td>-131</td>
<td>Wrong parameter type in the command string: long long int or long long int * expected</td>
</tr>
<tr>
<td>-132</td>
<td>Wrong parameter type in the command string: unsigned long long or unsigned long long * expected</td>
</tr>
<tr>
<td>-133</td>
<td>Error when creating actions tasks</td>
</tr>
<tr>
<td>-134</td>
<td>Changing the loop status is allowed in DISABLE state only</td>
</tr>
<tr>
<td>-135</td>
<td>Function is not allowed because group is not initialized or not referenced</td>
</tr>
<tr>
<td>-136</td>
<td>Wrong parameter type in the command string: charhex32 * expected</td>
</tr>
<tr>
<td>-137</td>
<td>Event&amp;Action: action threads number exceeds limit (must be ≤20)</td>
</tr>
<tr>
<td>-138</td>
<td>Event&amp;Action: action thread is running</td>
</tr>
<tr>
<td>-139</td>
<td>Event&amp;Action: Always event is not compatible with associated action</td>
</tr>
<tr>
<td>-200</td>
<td>Invalid socket</td>
</tr>
<tr>
<td>-201</td>
<td>The group is already in this mode</td>
</tr>
<tr>
<td>-202</td>
<td>Not allowed action due to an external motion interlock</td>
</tr>
<tr>
<td>-204</td>
<td>Function is not allowed because the feed forward is enabled</td>
</tr>
<tr>
<td>-205</td>
<td>Not enable in your configuration</td>
</tr>
<tr>
<td>-206</td>
<td>Dual encoder position error</td>
</tr>
<tr>
<td>-207</td>
<td>Gantry mode error: check Encoder Matrix and Decoupling Motor Matrix</td>
</tr>
<tr>
<td>-208</td>
<td>Not allowed action because piston is engaged</td>
</tr>
<tr>
<td>-209</td>
<td>INT board command failed: invalid card number or initialization not done</td>
</tr>
<tr>
<td>-210</td>
<td>Not allowed action due to (XStart ≤ Xangle ≤ XEnd) is not true</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-211</td>
<td>Not expected position after motion</td>
</tr>
<tr>
<td>-212</td>
<td>MagneticTrackPositionAtHome value is out of tolerance and can not be applied.</td>
</tr>
<tr>
<td>-1000</td>
<td>Zygo command execution failed</td>
</tr>
<tr>
<td>-1001</td>
<td>The controller is not connected to Zygo TCP server. Run ZygoStartInterferometer API.</td>
</tr>
<tr>
<td>-1002</td>
<td>Connection to Zygo TCP server failed</td>
</tr>
<tr>
<td>-1003</td>
<td>The XPS controller is already connected to Zygo TCP server</td>
</tr>
<tr>
<td>-1004</td>
<td>Zygo signal is not present</td>
</tr>
<tr>
<td>-1005</td>
<td>Zygo PEG configuration failed</td>
</tr>
<tr>
<td>-1006</td>
<td>Zygo error detected</td>
</tr>
</tbody>
</table>
# 8.10 Controller Status List

<table>
<thead>
<tr>
<th>Controller status code</th>
<th>code</th>
<th>Controller status description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROLLER_STATUS_OK</td>
<td>0x00000000</td>
<td>Controller status OK</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_INITIALIZATION_FAILED</td>
<td>0x00000001</td>
<td>Controller initialization failed</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_NB_OPENED_SOCKETS_REACHED_MAXIMUM_ALLOWED</td>
<td>0x00000002</td>
<td>Number of currently opened sockets reached maximum allowed number</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_CPU_OVERLOAD</td>
<td>0x00000004</td>
<td>Controller CPU is overloaded</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_CORRECTOR_OVER_CALCULATED</td>
<td>0x00000008</td>
<td>Current measured corrector calculation time exceeds the corrector period</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_PROFILER_OVER_CALCULATED</td>
<td>0x00000010</td>
<td>Profile generator calculating time exceeds ProfileGeneratorISRatio * IRSCorrectorPeriod</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_CORRECTOR_INTERRUPT_LOST</td>
<td>0x00000020</td>
<td>Controller has lost a corrector interrupt</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_INTERFERO_SIGNAL_NOT_PRESENT</td>
<td>0x00000040</td>
<td>Zygo interferometer signal is not present</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_INTERFERO_INITIALIZATION_FAILED</td>
<td>0x00000080</td>
<td>Zygo interferometer Ethernet initialisation failed</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_INTERFERO_ERROR_STATUS</td>
<td>0x00000100</td>
<td>Zygo interferometer error detected. Please check ZYGO Error Status</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_MOTION_VELOCITY_LIMITED</td>
<td>0x00000200</td>
<td>Motion velocity is limited</td>
</tr>
<tr>
<td>CONTROLLER_STATUS_LIFT_PIN_UP</td>
<td>0x00000400</td>
<td>Lift pin is UP</td>
</tr>
</tbody>
</table>

**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
9.0 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.

9.1 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:
9.2 Firmware Version Example

Open TCP connection

TCPOpen error?

Get firmware version

FirmwareVersionGet error?

Display XPS controller version

Close TCP socket

TCPClose error?

Display TCP close failed

Display TCP socket closed

Display TCP connection failed

Display error and close procedure
9.3 Gathering with Motion Example

Open TCP connection

Display TCP connection failed

YES TCPOpen error?

NO

Kill group

Display error and close procedure

YES

GroupKill error?

NO

Initialize group

Display error and close procedure

YES

GroupInitialize error?

NO

Home group

Display error and close procedure

YES

GroupHomeSearch error?

NO

Configure gathering

Display error and close procedure

YES

GatheringConfigurationSet error?

NO

Add event SGamma.MotionStart with action GatheringRun

Display error and close procedure

YES

EventAdd error?

NO Event occurred

Move positioner

Display error and close procedure

YES

GroupMoveRelative error?

NO
Stop gathering and save

Display error and close procedure

GatheringStopAndSave

Close TCP socket

Display TCP close failed

TCPClose

error?

NO

YES

Display TCP socket closed
9.4 External Gathering Example

Open TCP connection

Display TCP connection failed

YES

PTCPOpen error?

NO

Kill group

Display error and close procedure

YES

GroupKill error?

NO

Initialize group

Display error and close procedure

YES

GroupInitialize error?

NO

Home group

Display error and close procedure

YES

GroupHomeSearch error?

NO

Configure external gathering

Display error and close procedure

YES

GatheringExternalConfigurationSet error?

NO

Add event Immediate with action ExternalGatheringRun

Display error and close procedure

YES

EventAdd error?

NO

Wait end of external gathering

Get current number realized
9.5 Position Output Compare Example

Open TCP connection

Display TCP connection failed

TCOpen error?

YES

NO

Kill group

Display error and close procedure

GroupKill error?

YES

NO

Initialize group

Display error and close procedure

GroupInitialize error?

YES

NO

Home group

Display error and close procedure

GroupHomeSearch error?

YES

NO

Move positioner

Display error and close procedure

GroupMoveAbsolute error?

YES

NO

Set position compare parameters

Display error and close procedure

PositionerPositionCompareSet error?

YES

NO

Enable position compare mode

Display error and close procedure

PositionerPositionCompareEnable error?

YES

NO
9.6 Slave-Master Mode Example

Open TCP connection

Display TCP connection failed

TCPOpen error?

NO

Kill group

Display error and close procedure

GroupKill error?

NO

Initialize group

Display error and close procedure

GroupInitialize error?

NO

Home group

Display error and close procedure

GroupHomeSearch error?

NO

Set slave with its master

Display error and close procedure

SingleAxisSlaveParametersSet error?

NO

Enable slave-master mode

Display error and close procedure

SingleAxisSlaveModeEnable error?

NO

Move master positioner
9.7 Jogging Example

Open TCP connection

- Display TCP connection failed
  - YES
  - NO

- TCPOpen error?
  - YES
  - NO

- Kill group
  - YES
  - NO

- GroupKill error?
  - YES
  - NO

- Initialize group
  - YES
  - NO

- GroupInitialize error?
  - YES
  - NO

- Home group
  - YES
  - NO

- GroupHomeSearch error?
  - YES
  - NO

- Enable jog mode
  - YES
  - NO

- GroupJogModeEnable error?
  - YES
  - NO

- Set jog parameters to stop a positioner
  - YES
  - NO

- Set jog parameters to move a positioner
  - YES
  - NO
Display error and close

GroupJogParametersSet
error?

Display error and close

GroupJogModeDisable
error?

Close TCP

TCPClose
error?

Display TCP close failed

Display TCP socket closed
9.8 Tracking Example

Open TCP

Display TCP connection

YES  TCPOpe_n error?

NO  Kill

Display error and close

YES  GroupKill error?

NO  Initialize

Display error and close

YES  GroupInitialize error?

NO  Home

Display error and close

YES  GroupHomeSearch error?

NO  Set tracking

Display error and close

YES  PositionerAnalogTrackingPositionParameters Set

NO  Enable tracking

Display error and close

YES  GroupAnalogTrackingModeEnable

NO

Change the amplitude of analog

Disable tracking

Display error and close

YES  GroupAnalogTrackingModeDisable

NO  Close TCP

Display TCP close failed

YES  TCPClose error?

NO  Display TCP socket
9.9 Backlash

- Open TCP connection
  - Display TCP connection failed
    - YES TCPOpen error?
      - NO Kill group
        - YES Display error and close procedure
          - NO GroupKill error?
            - NO Enable Backlash
              - YES PositionerBacklashEnable error?
                - NO Initialize group
                  - YES GroupInitialize error?
                    - NO Home group
                      - YES GroupHomeSearch error?
                        - NO Modify Backlash value
                          - YES PositionerBacklash Set error?
                            - NO Move in positive direction

- Display error and close procedure
9.10 Timer Event and Global Variables

Open TCP connection 1

TCPOpen error?

NO

Calculate divisor

Configure timer

Display TCP connection failed

YES

Add event Timer with action ExecuteTCLScript

EventAdd error?

NO

Display error and close procedure

YES

Set global variable

GlobalArraySet error?

NO

Display error and close procedure

YES

Close TCP socket 1

Display TCP close failed

YES

TCPClose error?

NO

Display TCP socket 1 closed
9.11 Running Several Motion Processes Simultaneously

The groups must be initialized and homed before executing these processes.

```
Open TCP

Display TCP connection

YES

TCPOpen

error?

NO

Set Sgamma parameters of

YES

PositionerSGammaParametersSet

error?

NO

Set digital GPIO

YES

GPIODigitalSet

error?

NO

Set global

YES

GlobalArraySet

error?

NO

Execute TCL

YES

TCLScriptExecute

error?

NO

While in == 0

YES

Get digital GPIO

YES

GPIODigitalGet

error?

NO

If out == 0

YES

Set Sgamma parameters of

YES

PositionerSGammaParametersSet

error?

NO

Display error and close

NO

Open TCP

YES

Display TCP connection

NO

Set Sgamma parameters of

YES

PositionerSGammaParametersSet

error?

NO

Display error and close

NO
```
Service Form

Name: ___________________________  Return authorization #: ___________________________
Company: ____________________________________________  ___________________________
Address: ____________________________________________  ___________________________
Country: ____________________________________________  ___________________________
P.O. Number: ________________________________________  ___________________________
Item(s) Being Returned: ____________________________________________  ___________________________
Model#: ____________________________________________  Serial #: ___________________________

Description: _______________________________________________________________________________________

Reasons of return of goods (please list any specific problems): _______________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________

_____________________________________________________________________________________________
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