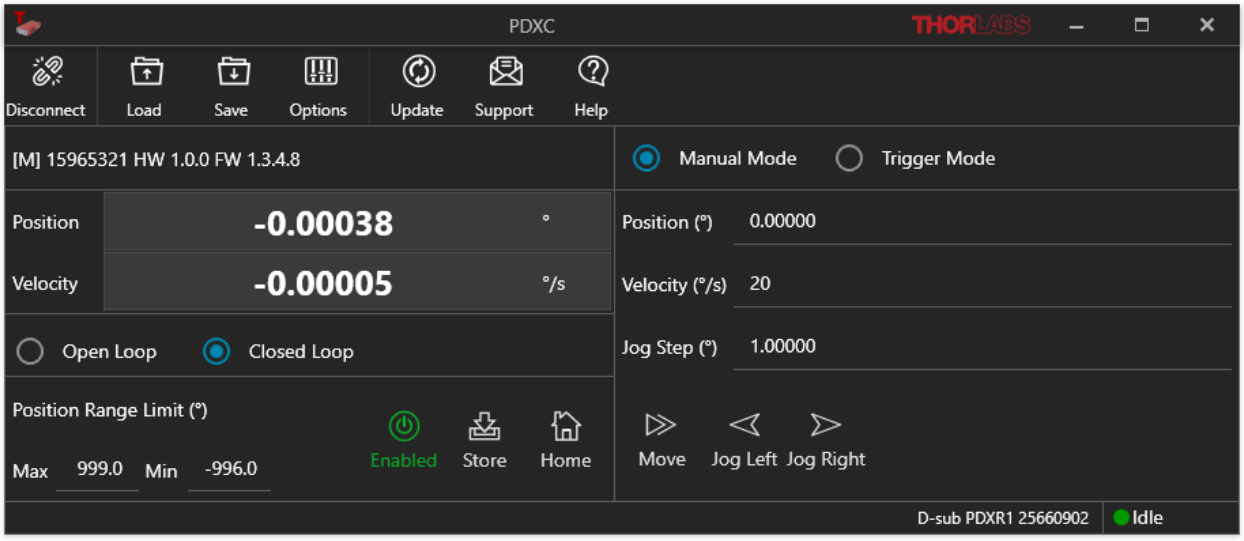
PDXC®
Piezo Stage Controller

SDK Manual

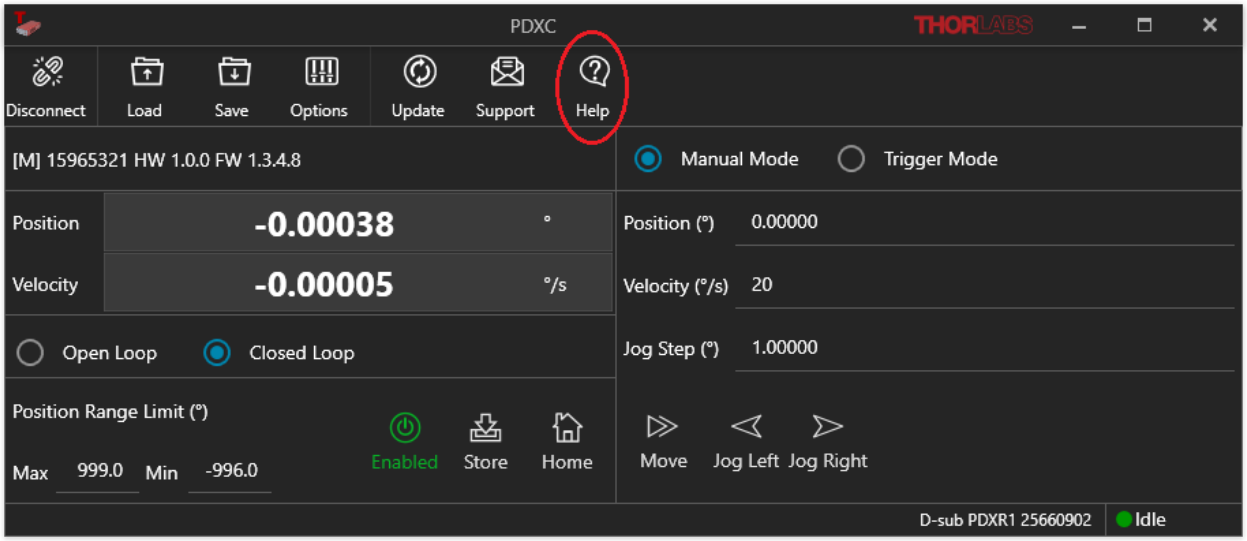


Copyright 2021-2023 Thorlabs, Inc.

Table of Contents

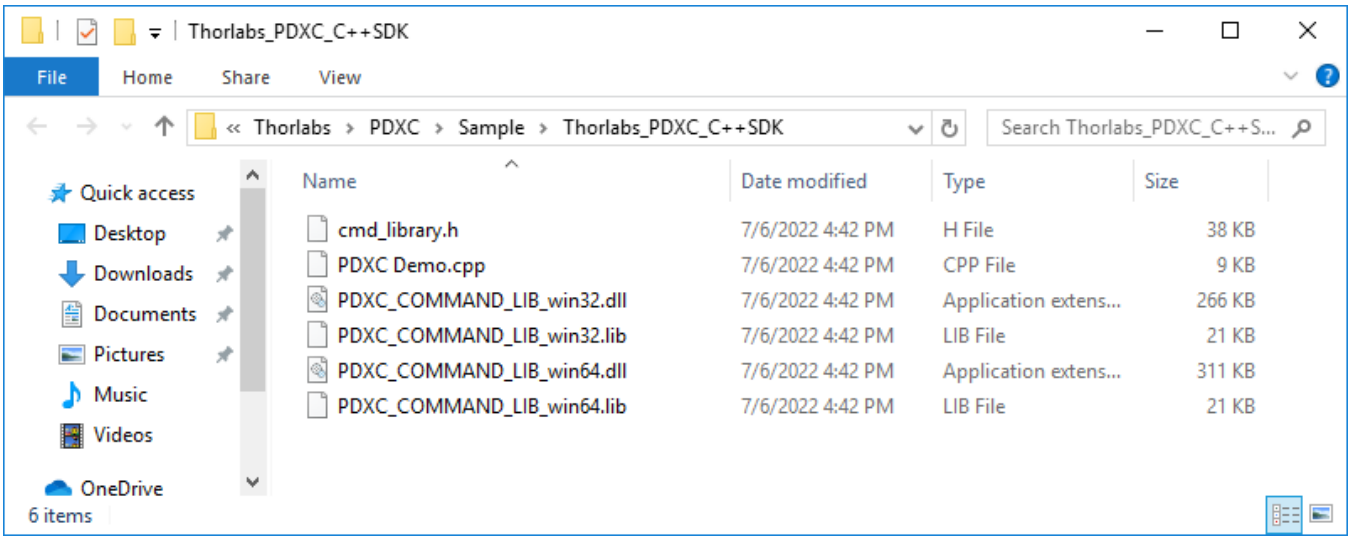
Chapter 1 Introduction

User can start software development in C/C++ develop environment, LabVIEW and Python.
The software development interface can be found by clicking *Help* in software menu.



Chapter 2 C++ Software Development Kit

The user can start software development with Visual Studio 2017 or later versions. The supported files are in **\Thorlabs_PDXC_C++SDK** under the **Sample** directory.



Copy **PDXC_COMMAND_LIB_win32.dll** or **PDXC_COMMAND_LIB_win64.dll** to your program folder, and make sure the library file(cmd_library.h) is in the same folder.

Below is the description of the header file **cmd_library.h**:

2.1. cmd_library.h File Reference

2.1.1. Functions

- **COMMANDLIB_API** int **List** (char *serialNo)
  list all the possible port on this computer.
- **COMMANDLIB_API** int **Open** (char *serialNo, int nBaud, int timeout)
  open port function.
- **COMMANDLIB_API** int **IsOpen** (char *serialNo)
  check opened status of port
- **COMMANDLIB_API** int **Close** (int hdl)
  close current opened port
- **COMMANDLIB_API** int **GetHandle** (char *serialNo)
  get handle of port
- **COMMANDLIB_API** int **Set** (int hdl, char *c, int var)
- **COMMANDLIB_API** int **Get** (int hdl, char *c, char *d)
- **COMMANDLIB_API** int **SetTimeout** (int hdl, int time)
- **COMMANDLIB_API** int **Get_Commands** (int hdl, int secondary, char *value)
- **COMMANDLIB_API** int **Get_CurrentStatus** (int hdl, int secondary, char *status)
  In D - sub Mode, it will return strings consist of ERR, KP, KI, KD, Current Loop, Velocity Level(open loop), Step size(open loop), Speed(Closed loop), Jog Step(closed loop), Home Flag and Abnormal Detection Status, parameters in

one command, results will be separated by ',' for each segment. In SMC Mode, it will return strings consist of ERR, Velocity and Step Size for Channel 1, Velocity and Step Size for channel 2.

- **COMMANDLIB_API** int **Get_ID** (int hdl, int secondary, char *ID)
  Get the model number, hardware and firmware versions
- **COMMANDLIB_API** int **Get_SN** (int hdl, int secondary, char *SN)
  Get the SN of PDXC controller.
- **COMMANDLIB_API** int **Get_SN2** (int hdl, int secondary, char *sn2)
  Get the Serial-Number followed by Item-Number of stage connected to PDXC controller, with the former one consist of strictly 8 bytes, and the latter one is not definte(e.g. when we get "SN024680PDX1/M", "SN024680" represent SN, "PDX1/M" represent Item-Number.
- **COMMANDLIB_API** int **Get_FV** (int hdl, int secondary, char *fv)
  Get strings, with former one is firmware version, the latter one is hardware version, the two are seperated by comma ','.
- **COMMANDLIB_API** int **Get_CalibrationIsCompleted** (int hdl, int secondary, char *homed)
  Get 'YES' when calibration complete, 'NO' when not.
- **COMMANDLIB_API** int **Get_DaisyChainStatus** (int hdl, int secondary, char *status)
  Get the current status, these are "Single-Mode", "Chain-Mode Main", "Chain-Mode Secondary1", "Chain-Mode Secondary2", .. "Chain-Mode Secondary8". Daisy - chain query command format is as following, M0:<CMD ? >_Sx : <CMD ? >_Sy : <CMD ? ><CR> M0 means the fixed Main, while Sx means x - th Secondary, x starts from 1 to 11. <CMD ? > means listed Query, eg.POS ? , and the return code is begin with "Sx:".
- **COMMANDLIB_API** int **Get_UserDataIsSaved** (int hdl, int secondary, char *saved)
  Get data saved status
- **COMMANDLIB_API** int **Get_KpOfPidParameters** (int hdl, int secondary, double *Kp)
  Get current Kp value
- **COMMANDLIB_API** int **Get_KiOfPidparameters** (int hdl, int secondary, double *Ki)
  Get current Ki value
- **COMMANDLIB_API** int **Get_KdOfPidparameters** (int hdl, int secondary, double *Kd)
  Get current Kd value
- **COMMANDLIB_API** int **Get_OpenLoopFrequency** (int hdl, int secondary, int *fry)
  Get current frequency of open loop of channel 1 in SMC mode.The unit is Hz.
- **COMMANDLIB_API** int **Get_OpenLoopFrequency2** (int hdl, int secondary, int *fry)
  Get current frequency of open loop of channel 2 in SMC mode.The unit is Hz.
- **COMMANDLIB_API** int **Get_OpenLoopFrequency3** (int hdl, int secondary, int *fry)
  Get current frequency of open loop for PD2/PD3.The unit is Hz.
- **COMMANDLIB_API** int **Get_LoopStatus** (int hdl, int secondary, int *loop)
  Get device loop status
- **COMMANDLIB_API** int **Get_AbnormalMoveDetect** (int hdl, int secondary, int *enable)
  Get whether abnormal move detect enable
- **COMMANDLIB_API** int **Get_ErrorMessage** (int hdl, int secondary, int *error)
  Get error message code
- **COMMANDLIB_API** int **Get_CurrentPosition** (int hdl, int secondary, double *position)
  Get current position counter.It only return postion value in D - SUB mode, other will return warnings
- **COMMANDLIB_API** int **Get_TargetTriggerPosition** (int hdl, int secondary, double *position)
  Get the target position which is calculated based on Analog In Gain and Analog In Offset.
- **COMMANDLIB_API** int **Get_Disabled** (int hdl, int secondary, int *disable)
  Get device disable state, under disable state stage will not move.
- **COMMANDLIB_API** int **Get_OpenLoopJogSize** (int hdl, int secondary, int *stepsize)
  Get the step size for open - loop for Channel 1
- **COMMANDLIB_API** int **Get_OpenLoopJogSize2** (int hdl, int secondary, int *value)

Get the step size for open - loop for Channel 2

- **COMMANDLIB_API** int **Get_OpenLoopJogSize3** (int hdl, int secondary, int *value)
  Get the step size for open - loop for PD2/PD3
- **COMMANDLIB_API** int **Get_ForwardAmplitude** (int hdl, int secondary, int *value)
  Get the forward amplitude for PD2/PD3/SMC
- **COMMANDLIB_API** int **Get_BackwardAmplitude** (int hdl, int secondary, int *value)
  Get the backward amplitude for PD2/PD3/SMC
- **COMMANDLIB_API** int **Get_SpeedStageType** (int hdl, int secondary, int *type)
  Get the current type of stage
- **COMMANDLIB_API** int **Get_AllParametersInExternalTrigger** (int hdl, int secondary, char *alltriggerState)
  Get all parameters except for Manual mode which has no parameters, they are "AR/AF,FR/FF[value],PR/PF[pos1],PR/PF[pos2]", which stands for Analog - In mode with rising / falling edge, Fixed - Size mode with rising / falling edge and value defined in PDX1(mm)/PDX2(mm)/PDXR(°) unit, and Two - Position - Switching mode with rising / falling for each postiion, assigned by pos1 and pos2.
- **COMMANDLIB_API** int **Get_CurrentStatusInExternalTrigger** (int hdl, int secondary, char *triggerState)
  Get the current status of external trigger mode, they are "ML"(Manual mode), "AR/AF"(Analog-In mode with rising/falling edge), "FR/FF[value]"(Fixed-Step size mode), and "PR/PF[pos1],PR/PF[pos2]"(Two-Postion Switching mode).
- **COMMANDLIB_API** int **Get_AnalogInputGain** (int hdl, int secondary, double *aiGain)
  Get gain value of analog input
- **COMMANDLIB_API** int **Get_AnalogInputOffSet** (int hdl, int secondary, double *aiOffSet)
  Get offset value of analog input
- **COMMANDLIB_API** int **Get_AnalogOutGain** (int hdl, int secondary, double *aoGain)
  Get gain value of analog output
- **COMMANDLIB_API** int **Get_AnalogOutOffSet** (int hdl, int secondary, double *aoOffSet)
  Get offset value of analog output
- **COMMANDLIB_API** int **Get_PositionLimit** (int hdl, int secondary, char *limit)
  Get values of positon limit by format of [Min, Max].
- **COMMANDLIB_API** int **Get_JoystickStatus** (int hdl, int secondary, int *num)
  Get the joystick status.
- **COMMANDLIB_API** int **Get_JoystickConfig** (int hdl, int secondary, char *value)
  Get the config of joystick
- **COMMANDLIB_API** int **Set_DaisyChain** (int hdl, int index)
  Set device position in daisy-chain.
- **COMMANDLIB_API** int **Set_TargetSpeed** (int hdl, int secondary, int speed)
  Set the desired speed Work only under D- SUB mode, other will return warnings.
- **COMMANDLIB_API** int **Set_OpenLoopFrequency** (int hdl, int secondary, int value)
  Set the open loop frequency of channel 1.The unit is Hz. Must in SMC mode
- **COMMANDLIB_API** int **Set_OpenLoopFrequency2** (int hdl, int secondary, int value)
  Set the open loop frequency of channel 2.The unit is Hz. Must in SMC mode
- **COMMANDLIB_API** int **Set_OpenLoopFrequency3** (int hdl, int secondary, int value)
  Set the output frequency of D-Sub stage without encoder(PD2/PD3).The unit is Hz.
- **COMMANDLIB_API** int **Set_OpenLoopJogSize** (int hdl, int secondary, int stepsize)
  Set the step size for open - loop for Channel 1
- **COMMANDLIB_API** int **Set_OpenLoopJogSize2** (int hdl, int secondary, int stepsize)
  Set the step size for open - loop for Channel 2
- **COMMANDLIB_API** int **Set_OpenLoopJogSize3** (int hdl, int secondary, int stepsize)
  Set the step size for open - loop for PD2/PD3

- **COMMANDLIB_API** int **Set_ForwardAmplitude** (int hdl, int secondary, int value)
  Set the forward amplitude for PD2/PD3/SMC
- **COMMANDLIB_API** int **Set_BackwardAmplitude** (int hdl, int secondary, int value)
  Set the backward amplitude for PD2/PD3/SMC
- **COMMANDLIB_API** int **Set_PositionCalibration** (int hdl, int secondary, int home)
  Calibrate the QDEC counter after power - up.
- **COMMANDLIB_API** int **Set_AbnormalMoveDetect** (int hdl, int secondary, int enable)
  Switch on or off the abnormal move detection, default is on.Used to detect stage stuck or move by external force.
- **COMMANDLIB_API** int **Set_Loop** (int hdl, int secondary, int loop)
  Switch closeloop and open loop
- **COMMANDLIB_API** int **Set_StepPulseAndResponse** (int hdl, int secondary, double value)
  Set step position
- **COMMANDLIB_API** int **Set_TargetPosition** (int hdl, int secondary, double position)
  Set the target position counter.Only work in Manual Mode. In Open Loop, will use continuous pulses at preset amplitude until reach the destination without PID control(including Speed and Acurate position). Work in both Open and Closed Loop in D - SUB mode only. While in Closed Loop, will add PID control, and anlaog move near destination.
- **COMMANDLIB_API** int **Set_KpOfPidParameters** (int hdl, int secondary, double Kp)
  Set current Kp value, will store in Flash memory.
- **COMMANDLIB_API** int **Set_KiOfPidParameters** (int hdl, int secondary, double Ki)
  Set current Ki value, will store in Flash memory.
- **COMMANDLIB_API** int **Set_KdOfPidParameters** (int hdl, int secondary, double Kd)
  Set current Kd value, will store in Flash memory.
- **COMMANDLIB_API** int **Set_AnalogInputGain** (int hdl, int secondary, double aiGain)
  *Change gain value of analog input, default input voltage range is[-10V, 10V], standing for PDX1[+ -10mm]/ PDX2[+ - 2.5mm]/PDXR[+ - 180°] position range.If change value of Gain&Offset, we will get new input voltage range.The analog input gain vaule(ING), calculated by(max - min) / 20, analog input offSet value(INO) calculated by(min + max) / 2. E.g.If new wanted input range is in[0V, 5V], which means analog input gain is 0.25, analog input offSet value is 2.5. (the new Voltage Sampled is function of Vold, ING and INO Vnew = ING * Vold + 1.5(1 - ING) - 0.15*INO, where Vout is default range's output.)*
- **COMMANDLIB_API** int **Set_AnalogInputOffSet** (int hdl, int secondary, double aiOffset)
  *Change offset value of analog input, default input voltage range is[-10V, 10V].If change value of Gain&Offset, we will get new input voltage range.The analog input gain vaule(ING), calculated by(max - min) / 20, analog input offSet value(INO) calculated by(min + max) / 2. E.g.If new wanted input range is in[0V, 5V], which means analog input gain is 0.25, analog input offSet value is 2.5. (the new Voltage Sampled is function of Vold, ING and INO Vnew = ING * Vold + 1.5(1 - ING) - 0.15*INO, where Vout is default range's output.)*
- **COMMANDLIB_API** int **Set_AnalogOutGain** (int hdl, int secondary, double aoGain)
  *Change gain Value of analog output, default output range is[-10V, 10V].If change value of Gain&Offset, we will get new output range.The analog out gain value(OUG), The analog out offset value(OUO).E.g.If new wanted output range is in[0V, 5V], which means OUG = 0.25, OUO = 2.5. (the new voltage output for DAC is function of position, which equals Vdac_new = (Vdac_old - 1.5)*OUG + OUO / 6.8 + 1.5, where Vdac_old = (pos / 100000 + 10) / 20 * 3).*
- **COMMANDLIB_API** int **Set_AnalogOutOffSet** (int hdl, int secondary, double aoOffset)
  *Change offset Value of analog output, default output range is[-10V, 10V].If change value of Gain&Offset, we will get new output range.The analog out gain value(OUG), The analog out offset value(OUO).E.g.If new wanted output range is in[0V, 5V], which means OUG = 0.25, OUO = 2.5. (the new voltage output for DAC is function of position, which equals Vdac_new = (Vdac_old - 1.5)*OUG + OUO / 6.8 + 1.5, where Vdac_old = (pos / 100000 + 10) / 20 * 3).*
- **COMMANDLIB_API** int **Set_AllCustomerData** (int hdl, int secondary, int saveState)
  Data contains SMC data and non - SMC data, the former one contains velocity and step size for each channel, and the later one contains status(reserved), speed for closedloop, Velocity Level for openloop, position and step size, daisy - chain number, input trigger value abnormal detection value input / output gain and offset(4 in all), and step distance.
- **COMMANDLIB_API** int **Set_OpenLoopMoveForward** (int hdl, int secondary, int pulses, int channel)

Set Open Loop Move Forward

- **COMMANDLIB_API** int **Set_OpenLoopMoveBack** (int hdl, int secondary, int pulses, int channel)
  Set Open Loop Move Backward
- **COMMANDLIB_API** int **Set_Disabled** (int hdl, int secondary, int disable)
  Set controller disabled
- **COMMANDLIB_API** int **Set_CurrentStatusInExternalTrigger** (int hdl, int secondary, char *triggerMode)
  Set the current status of external trigger mode
- **COMMANDLIB_API** int **Set_PositionLimit** (int hdl, int secondary, double min, double max)
  Set the minimum and maximum values of position, the controller will not respond to the value surpass this range.
- **COMMANDLIB_API** int **Set_JoystickConfig** (int hdl, int secondary, int value, int value2)
  Set the joystick config for device.
- **COMMANDLIB_API** int **Set_AllHome** (int hdl, char *indexs)
  Set all devices home.
- **COMMANDLIB_API** int **Set_AllStore** (int hdl, char *indexs, int value)
  Save/Erase data for all devices.

2.1.2. Function Documentation

COMMANDLIB_API int Close (int **hdl***)*

close current opened port

Parameters

hdl	handle of port.

Returns

0: success; negative number: failed.

COMMANDLIB_API int Get (int **hdl***, char **** **c***, char **** **d***)*

COMMANDLIB_API int Get_AbnormalMoveDetect (int **hdl***, int* **secondary***, int **** **enable***)*

Get whether abnormal move detect enable

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
enable	('1' : enabled, '0' : disabled)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_AllParametersInExternalTrigger (int **hdl***, int* **secondary***, char **** **alltriggerState***)*

Get all parameters except for Manual mode which has no parameters, they are "AR/AF,FR/FF[value],PR/PF[pos1],PR/PF[pos2]", which stands for Analog - In mode with rising / falling edge, Fixed - Size mode with rising / falling edge and value defined in PDX1(mm)/PDX2(mm)/PDXR(°) unit, and Two - Position - Switching mode with rising / falling for each postiion, assigned by pos1 and pos2.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
alltriggerState	device all trigger State

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_AnalogInputGain (int* **hdl**, int* **secondary**, double ** **aiGain**)

Get gain value of analog input

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
aiGain	analog input gain :PDX1/PDX2:[0.1,1]; PDXR1:[0.1,100]

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_AnalogInputOffSet (int* **hdl**, int* **secondary**, double ** **aiOffSet**)

Get offset value of analog input

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
aiOffSet	analog input offset: PDX1:[-10,10]mm; PDX2:[-2.5, 2.5]mm; PDXR:[-999.9,999.9]°

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_AnalogOutGain (int* **hdl**, int* **secondary**, double ** **aoGain**)

Get gain value of analog output

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
aoGain	analog output gain[0.1,1]

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_AnalogOutOffSet (int* **hdl**, int* **secondary**, double ** **aoOffSet**)

Get offset value of analog output

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
aoOffSet	analog output offset: PDX1/PDX2:[-10,10]mm; PDXR:[-999.9,999.9]°

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_BackwardAmplitude (int* **hdl**, int* **secondary**, int ** **value**)

Get the backward amplitude for PD2/PD3/SMC

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	backward amplitude (10~100)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_CalibrationIsCompleted (int hdl, int secondary, char * homed)

Get 'YES' when calibration complete, 'NO' when not.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
homed	device calibration completed status

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_Commands (int hdl, int secondary, char * value)

COMMANDLIB_API int Get_CurrentPosition (int hdl, int secondary, double * position)

Get current position counter.It only return postion value in D - SUB mode, other will return warnings

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
position	current position: PDX1[-10,10]mm; PDX2[-2.5,2.5]mm; PDXR:[-999.9,999.9]

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_CurrentStatus (int hdl, int secondary, char * status)

In D - sub Mode, it will return strings consist of ERR, KP, KI, KD, Current Loop, Velocity Level(open loop), Step size(open loop), Speed(Closed loop), Jog Step(closed loop), Home Flag and Abnormal Detection Status, parameters in one command, results will be separated by ',' for each segment. In SMC Mode, it will return strings consist of ERR, Velocity and Step Size for Channel 1, Velocity and Step Size for channel 2.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
status	device status

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_CurrentStatusInExternalTrigger (int hdl, int secondary, char * triggerState)

Get the current status of external trigger mode, they are "ML"(Manual mode), "AR/AF"(Analog-In mode with rising/falling edge), "FR/FF[value]"(Fixed-Step size mode), and "PR/PF[pos1],PR/PF[pos2]"(Two-Postion Switching mode).

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
triggerState	status of external trigger mode

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_DaisyChainStatus (int* hdl*, int* secondary*, char ** status*)

Get the current status, these are "Single-Mode", "Chain-Mode Main", "Chain-Mode Secondary1", "Chain-Mode Secondary2", .. "Chain-Mode Secondary8". Daisy - chain query command format is as following, M0:<CMD ? >_Sx : <CMD ? >_Sy : <CMD ? ><CR> M0 means the fixed Main, while Sx means x - th Secondary, x starts from 1 to 11. <CMD ? > means listed Query, eg.POS ? , and the return code is begin with "Sx:".

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
status	current Daisy - chain status

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_Disabled (int* hdl*, int* secondary*, int ** disable*)

Get device disable state, under disable state stage will not move.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
disable	disable state(0 :enabled, 1 :disabled)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_ErrorMessage (int* hdl*, int* secondary*, int ** error*)

Get error message code

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
error	error code (0 : None errors occurred, 1 : command not defined, 2 : command Data out - of - range, 3 : device failed to execute last command, 4 : no waveform data loaded, 5 : Stage need Home first, 6 : device works in the wrong mode, 7 : stage move abnormal, 8 : excessive current occurred, 9 : over temperature occurred, 17 : unkown error occurred)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_ForwardAmplitude (int* hdl*, int* secondary*, int ** value*)

Get the forward amplitude for PD2/PD3/SMC

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	forward amplitude (10~100)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_FV (int hdl, int secondary, char * fv)

Get strings, with former one is firmware version, the latter one is hardware version, the two are seperated by comma ','.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
fv	firmware version and hardware version

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_ID (int hdl, int secondary, char * ID)

Get the model number, hardware and firmware versions

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
ID	device id string

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_JoystickConfig (int hdl, int secondary, char * value)

Get the config of joystick

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main
value	Return the array of X,Y,x=0 means no device, x=1 means Single-Mode or Main, x=2 to 12 means Secondary1 to Secondary11, y means the number of knob on joystick(from 1 to n),each group of data is separated by space such as "[1,1 2,2 4,3]".

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_JoystickStatus (int hdl, int secondary, int * num)

Get the joystick status.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main
num	The number of knob on joystick. 0: No joystick is connected to the

	device

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_KdOfPidparameters (int* **hdl**, int* **secondary**, double* * **Kd**)

Get current Kd value

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
Kd	Kd of PID

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_KiOfPidparameters (int* **hdl**, int* **secondary**, double* * **Ki**)

Get current Ki value

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
Ki	Ki of PID

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_KpOfPidParameters (int* **hdl**, int* **secondary**, double* * **Kp**)

Get current Kp value

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
Kp	Kp of PID

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_LoopStatus (int* **hdl**, int* **secondary**, int* * **loop**)

Get device loop status

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
loop	loop status(1 :open loop, 0 :close loop)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_OpenLoopFrequency (int* **hdl**, int* **secondary**, int* * **fry**)

Get current frequency of open loop of channel 1 in SMC mode.The unit is Hz.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
fry	current frequency of open loop

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_OpenLoopFrequency2 (int hdl, int secondary, int * fry)

Get current frequency of open loop of channel 2 in SMC mode.The unit is Hz.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
fry	current frequency of open loop

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_OpenLoopFrequency3 (int hdl, int secondary, int * fry)

Get current frequency of open loop for PD2/PD3.The unit is Hz.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
fry	current frequency of open loop

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_OpenLoopJogSize (int hdl, int secondary, int * stepsize)

Get the step size for open - loop for Channel 1

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
stepsize	open loop jog step size (1~65535)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_OpenLoopJogSize2 (int hdl, int secondary, int * value)

Get the step size for open - loop for Channel 2

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	open loop jog step size (1~65535)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_OpenLoopJogSize3 (int* **hdl**, int* **secondary**, int * * **value**)

Get the step size for open - loop for PD2/PD3

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	open loop jog step size (1~400000)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_PositionLimit (int* **hdl**, int* **secondary**, char * * **limit**)

Get values of positon limit by format of [Min, Max].

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
limit	Positon limit. Format of [Min, Max]

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_SN (int* **hdl**, int* **secondary**, char * * **SN**)

Get the SN of PDXC controller.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
SN	PDXC controller SN

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_SN2 (int* **hdl**, int* **secondary**, char * * **sn2**)

Get the Serial-Number followed by Item-Number of stage connected to PDXC controller, with the former one consist of strictly 8 bytes, and the latter one is not definte(e.g. when we get "SN024680PDX1/M", "SN024680" represent SN, "PDX1/M" represent Item-Number.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
sn2	Serial-Number and Item-Number of stage

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_SpeedStageType (int* **hdl**, int* **secondary**, int * * **type**)

Get the current type of stage

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
type	stage type(0 :PDX1/PDX2/PDXR stage, 1 :SMC stage,2 :PD2/PD3 stage)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_TargetTriggerPosition (int* hdl*, int* secondary*, double ** position*)

Get the target position which is calculated based on Analog In Gain and Analog In Offset.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
position	target position: PDX1[-10,10]mm; PDX2[-2.5,2.5]mm; PDXR:[-999.9,999.9]

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Get_UserDataIsSaved (int* hdl*, int* secondary*, char ** saved*)

Get data saved status

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
saved	saved status(bit 0 : close - loop stage, bit 1 : SMC stages [0 : no user data at all, 1 : only close - loop stage , 2 : only SMC stages , 3 : both close - loop and SMC stages.])

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int GetHandle (char ** serialNo*)

get handle of port

Parameters

serialNo	serial number of the device to be checked.

Returns

-1:no handle non-negtive number: handle.

COMMANDLIB_API int IsOpen (char ** serialNo*)

check opened status of port

Parameters

serialNo	serial number of the device to be checked.

Returns

0: port is not opened; 1: port is opened.

COMMANDLIB_API int List (char * serialNo)

list all the possible port on this computer.

Parameters

serialNo	port list returned string include serial number and device descriptor, separated by comma

Returns

non-negtive number: number of device in the list; negtive number : failed.

COMMANDLIB_API int Open (char * serialNo, int nBaud, int timeout)

open port function.

Parameters

serialNo	serial number of the device to be opened, use GetPorts function to get exist list first.
nBaud	bit per second of port
timeout	set timeout value in (s)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set (int hdl, char * c, int var)

COMMANDLIB_API int Set_AbnormalMoveDetect (int hdl, int secondary, int enable)

Switch on or off the abnormal move detection, default is on.Used to detect stage stuck or move by external force.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
enable	detect enable(1 : Enable , 0 :Disable)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_AllCustomerData (int hdl, int secondary, int saveState)

Data contains SMC data and non - SMC data, the former one contains velocity and step size for each channel, and the later one contains status(reserved), speed for closedloop, Velocity Level for openloop, position and step size, daisy - chain number, input trigger value abnormal detection value input / output gain and offset(4 in all), and step distance.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
saveState	set to save or erase(1:save data;0:erase data.)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_AllHome (int hdl, char * indexs)

Set all devices home.

Parameters

hdl	handle of port.
indexs	Index list of all devices(0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11[separated by ',']

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_AllStore (int* hdl*, char** indexs*, int* value*)

Save/Erase data for all devices.

Parameters

hdl	handle of port.
indexs	Index list of all devices(0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11[separated by ',']
value	1:Save data; 0:Erase data

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_AnalogInputGain (int* hdl*, int* secondary*, double* aiGain*)

Change gain value of analog input, default input voltage range is[-10V, 10V], standing for PDX1[+ -10mm]/ PDX2[+ -2.5mm]/PDXR[+ - 10°] position range.If change value of Gain&Offset, we will get new input voltage range.The analog input gain vaule(ING), calculated by(max - min) / 20, analog input offSet value(INO) calculated by(min + max) / 2. E.g.If new wanted input range is in[0V, 5V], which means analog input gain is 0.25, analog input offSet value is 2.5. (the new Voltage Sampled is function of Vold, ING and INO Vnew = ING * Vold + 1.5(1 - ING) - 0.15*INO, where Vout is default range's output.)

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
aiGain	gain value of analog input: PDX1/PDX2:[0.1,1]; PDXR1:[0.1,100]

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_AnalogInputOffSet (int* hdl*, int* secondary*, double* aiOffset*)

Change offset value of analog input, default input voltage range is[-10V, 10V].If change value of Gain&Offset, we will get new input voltage range.The analog input gain vaule(ING), calculated by(max - min) / 20, analog input offSet value(INO) calculated by(min + max) / 2. E.g.If new wanted input range is in[0V, 5V], which means analog input gain is 0.25, analog input offSet value is 2.5. (the new Voltage Sampled is function of Vold, ING and INO Vnew = ING * Vold + 1.5(1 - ING) - 0.15*INO, where Vout is default range's output.)

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
aiOffset	offset value of analog input: PDX1:[-10,10]mm; PDX2:[-2.5, 2.5]mm; PDXR:[-999.9,999.9]°

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_AnalogOutGain (int* **hdl**, int* **secondary**, double* **aoGain**)

Change gain Value of analog output, default output range is[-10V, 10V].If change value of Gain&Offset, we will get new output range.The analog out gain value(OUG), The analog out offset value(OUO).E.g.If new wanted output range is in[0V, 5V], which means OUG = 0.25, OUO = 2.5. (the new voltage output for DAC is function of position, which equals Vdac_new = (Vdac_old - 1.5)*OUG + OUO / 6.8 + 1.5, where Vdac_old = (pos / 100000 + 10) / 20 * 3).

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
aoGain	gain Value of analog output:[0.1,1]

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_AnalogOutOffSet (int* **hdl**, int* **secondary**, double* **aoOffset**)

Change offset Value of analog output, default output range is[-10V, 10V].If change value of Gain&Offset, we will get new output range.The analog out gain value(OUG), The analog out offset value(OUO).E.g.If new wanted output range is in[0V, 5V], which means OUG = 0.25, OUO = 2.5. (the new voltage output for DAC is function of position, which equals Vdac_new = (Vdac_old - 1.5)*OUG + OUO / 6.8 + 1.5, where Vdac_old = (pos / 100000 + 10) / 20 * 3).

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
aoOffset	offset Value of analog output: PDX1/PDX2:[-10,10]mm; PDXR:[-999.9,999.9]°

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_BackwardAmplitude (int* **hdl**, int* **secondary**, int* **value**)

Set the backward amplitude for PD2/PD3/SMC

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	backward amplitude (10~100)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_CurrentStatusInExternalTrigger (int* **hdl**, int* **secondary**, char * **triggerMode**)

Set the current status of external trigger mode

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
triggerMode	"ML"(Manual mode), "AR/AF"(Analog - In mode with rising / falling edge), "FR/FF[value]"(Fixed - Step size mode), and "PR/PF[pos1],PR/PF[pos2]"(Two - Postion Switching mode).

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_DaisyChain (int* **hdl**, int* **index**)

Set device position in daisy-chain.

Parameters

hdl	handle of port.
index	index in daisy chain (0:Single-Mode, 1:Main, 2 -12 : Secondary1 - Secondary11)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_Disabled (int* **hdl**, int* **secondary**, int* **disable**)

Set controller disabled

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
disable	disable state(0:enable the device, 1:disable the device)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_ForwardAmplitude (int* **hdl**, int* **secondary**, int* **value**)

Set the forward amplitude for PD2/PD3/SMC

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	forward amplitude (10~100)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_JoystickConfig (int* **hdl**, int* **secondary**, int* **value**, int* **value2**)

Set the joystick config for device.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main
value	0:No device; 1:Single-Mode or Main; 2 -12 : Secondary1-Secondary11
value2	The number of knob on joystick[from 0 to n(0:no knob; 1:fixed to 1:Single-Mode or Main)]

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_KdOfPidParameters (int* **hdl**, int* **secondary**, double* **Kd**)

Set current Kd value, will store in Flash memory.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
Kd	Kd of PID

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_KiOfPidParameters (int* hdl*, int* secondary*, double* Ki*)

Set current Ki value, will store in Flash memory.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
Ki	Ki of PID

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_KpOfPidParameters (int* hdl*, int* secondary*, double* Kp*)

Set current Kp value, will store in Flash memory.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
Kp	Kp of PID

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_Loop (int* hdl*, int* secondary*, int* loop*)

Switch closeloop and open loop

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
loop	loop type (1 : open loop (default), 0 : close loop.)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_OpenLoopFrequency (int* hdl*, int* secondary*, int* value*)

Set the open loop frequency of channel 1.The unit is Hz. Must in SMC mode

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	frequency of open loop

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_OpenLoopFrequency2 (int hdl, int secondary, int value)

Set the open loop frequency of channel 2.The unit is Hz. Must in SMC mode

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	frequency of open loop

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_OpenLoopFrequency3 (int hdl, int secondary, int value)

Set the output frequency of D-Sub stage without encoder(PD2/PD3).The unit is Hz.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	frequency

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_OpenLoopJogSize (int hdl, int secondary, int stepsize)

Set the step size for open - loop for Channel 1

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
stepsize	open loop jog step size (1~65535)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_OpenLoopJogSize2 (int hdl, int secondary, int stepsize)

Set the step size for open - loop for Channel 2

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
stepsize	open loop jog step size (1~65535)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_OpenLoopJogSize3 (int hdl, int secondary, int stepsize)

Set the step size for open - loop for PD2/PD3

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11

stepsize	open loop jog step size (1~400000)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_OpenLoopMoveBack (int* hdl*, int* secondary*, int* pulses*, int* channel*)

Set Open Loop Move Backward

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
pulses	pulses of move channel:SMC[1,65535]; PD2/PD3[1,400000]
channel	Move backward channel (0 : channel 1, 1 : channel 2, others :both channels)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_OpenLoopMoveForward (int* hdl*, int* secondary*, int* pulses*, int* channel*)

Set Open Loop Move Forward

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
pulses	pulses of move channel:SMC[1,65535]; PD2/PD3[1,400000]
channel	Move fordward channel (0 : channel 1, 1 : channel 2, others :both channels)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_PositionCalibration (int* hdl*, int* secondary*, int* home*)

Calibrate the QDEC counter after power - up.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
home	controller home(1: Yes, 0:No)

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_PositionLimit (int* hdl*, int* secondary*, double* min*, double* max*)

Set the minimum and maximum values of position, the controller will not respond to the value surpass this range.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
min	Minimum value of position, PDX1 range[-10,max] mm; PDX2 range[-2.5,max] mm; PDXR range[-999.9,max] °
max	Maximum value of position, PDX1 range[min,10] mm; PDX2 range[min,2.5] mm;PDXR range[min.9,999.9] °

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_StepPulseAndResponse (int* **hdl**, *int* **secondary**, *double* **value**)

Set step position

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
value	relative distance to be moved:PDX1:[-10,-0.00001]mm,[0.00001,10]mm; PDX2: [-2.5, -0.0003]mm, [0.0003, 2.5]mm; PDXR:[-180,-0.00005] °, [0.00005,180] °

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_TargetPosition (int* **hdl**, *int* **secondary**, *double* **position**)

Set the target position counter.Only work in Manual Mode. In Open Loop, will use continuous pulses at preset amplitude until reach the destination without PID control(including Speed and Acurate position). Work in both Open and Closed Loop in D - SUB mode only. While in Closed Loop, will add PID control, and anlaog move near destination.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
position	Target Position: PDX1[-10,10]mm; PDX2[-2.5,2.5]mm; PDXR:[-180,180]°

Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int Set_TargetSpeed (int* **hdl**, *int* **secondary**, *int* **speed**)

Set the desired speed Work only under D- SUB mode, other will return warnings.

Parameters

hdl	handle of port.
secondary	0:Single-Mode or Main; 1 -11 : Secondary1-Secondary11
speed	desired speed(PDX1:2-20 mm/s; PDX2:1-10 mm/s; PDXR:10-30 °/s)

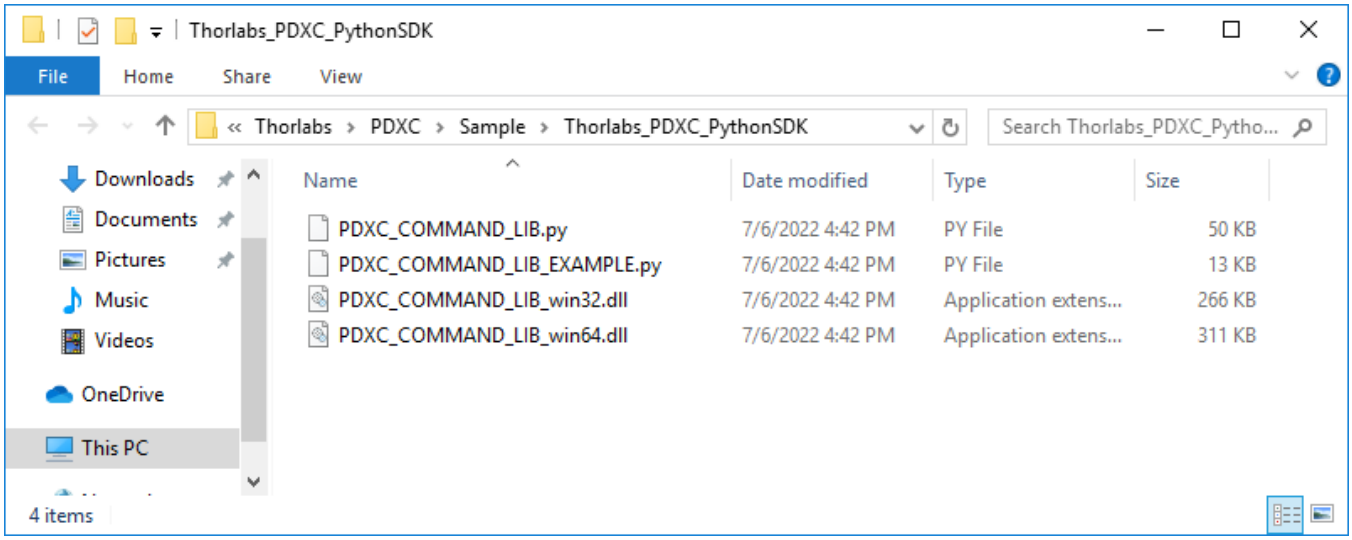
Returns

non-negtive number: hdl number returned successfully; negtive number : failed.

COMMANDLIB_API int SetTimeout (int* **hdl**, *int* **time**)

Chapter 3 Python Software Development Kit

The user can start software development with Python 3.7 or later versions. The supported files are in **\Thorlabs_PDXC_PythonSDK** under the **Sample** directory.



User can import **PDXC_COMMAND_LIB.py** to your python project, that's the wrapper for **PDXC_COMMAND_LIB(PDXC_COMMAND_LIB_win32.dll** in C/C++ development environment). Copy **PDXC_COMMAND_LIB_win32.dll(for 32-bit application)** to your program folder, and make sure the library file and **PDXC_COMMAND_LIB.py** file are in the same folder. The **PDXC_COMMAND_LIB_EXAMPLE.py** is the example code for how to use the python APIs.

User can also replace the reference win32 lib to x64 lib for 64-bit application and modify the **__init__** fuction code " **lib_path = "./PDXC_COMMAND_LIB_win32.dll"**" to " **lib_path = "./PDXC_COMMAND_LIB_win64.dll"**" in **PDXC_COMMAND_LIB.py** file.

3.1. PDXC_COMMAND_LIB Namespace Reference

3.1.1. pdxc Class Reference

Public Member Functions

- def **__init__** (self)
- def **Open** (self, serialNo, nBaud, timeout)
- def **IsOpen** (self, serialNo)
- def **GetHandle** (self, serialNo)
- def **Close** (self)
- def **GetCurrentStatus** (self, secondary, status)
- def **GetSN** (self, secondary, SN)
- def **GetSN2** (self, secondary, SN2)
- def **GetFV** (self, secondary, fv)
- def **GetCalibrationIsCompleted** (self, secondary, homed)
- def **GetDaisyChainStatus** (self, secondary, status)
- def **GetUserDataIsSaved** (self, secondary, saved)
- def **GetKpOfPidParameters** (self, secondary, Kp)

- def **GetKiOfPidParameters** (self, secondary, Ki)
- def **GetKdOfPidParameters** (self, secondary, Kd)
- def **GetOpenLoopFrequency** (self, secondary, fry)
- def **GetOpenLoopFrequency2** (self, secondary, fry2)
- def **GetOpenLoopFrequency3** (self, secondary, fry3)
- def **GetLoopStatus** (self, secondary, loop)
- def **GetAbnormalMoveDetect** (self, secondary, enable)
- def **GetErrorMessage** (self, secondary, error)
- def **GetCurrentPosition** (self, secondary, position)
- def **GetTargetTriggerPosition** (self, secondary, position)
- def **GetDisabled** (self, secondary, disable)
- def **GetOpenLoopJogSize** (self, secondary, stepsize)
- def **GetOpenLoopJogSize2** (self, secondary, stepsize2)
- def **GetOpenLoopJogSize3** (self, secondary, stepsize3)
- def **GetForwardAmplitude** (self, secondary, forampli)
- def **GetBackwardAmplitude** (self, secondary, baampli)
- def **GetSpeedStageType** (self, secondary, type)
- def **GetAllParametersInExternalTrigger** (self, secondary, alltriggerState)
- def **GetCurrentStatusInExternalTrigger** (self, secondary, triggerState)
- def **GetAnalogInputGain** (self, secondary, aiGain)
- def **GetAnalogInputOffSet** (self, secondary, aiOffSet)
- def **GetAnalogOutGain** (self, secondary, aoGain)
- def **GetAnalogOutOffSet** (self, secondary, aoOffSet)
- def **GetPositionLimit** (self, secondary, limit)
- def **GetJoystickStatus** (self, secondary, number)
- def **GetJoystickConfig** (self, secondary, value)
- def **SetDaisyChain** (self, index)
- def **SetTargetSpeed** (self, secondary, speed)
- def **SetOpenLoopFrequency** (self, secondary, fry)
- def **SetOpenLoopFrequency2** (self, secondary, fry2)
- def **SetOpenLoopFrequency3** (self, secondary, fry3)
- def **SetOpenLoopJogSize** (self, secondary, stepsize)
- def **SetOpenLoopJogSize2** (self, secondary, stepsize2)
- def **SetOpenLoopJogSize3** (self, secondary, stepsize3)
- def **SetForwardAmplitude** (self, secondary, forampli)
- def **SetBackwardAmplitude** (self, secondary, baampli)
- def **SetPositionCalibration** (self, secondary, home)
- def **SetAbnormalMoveDetect** (self, secondary, enable)
- def **SetLoop** (self, secondary, loop)
- def **SetTargetPosition** (self, secondary, position)
- def **SetKpOfPidParameters** (self, secondary, Kp)
- def **SetKiOfPidParameters** (self, secondary, Ki)
- def **SetKdOfPidParameters** (self, secondary, Kd)
- def **SetAnalogInputGain** (self, secondary, aiGain)
- def **SetAnalogInputOffSet** (self, secondary, aiOffset)
- def **SetAnalogOutGain** (self, secondary, aoGain)
- def **SetAnalogOutOffSet** (self, secondary, aoOffset)
- def **SetAllCustomerData** (self, secondary, saveState)
- def **SetOpenLoopMoveForward** (self, secondary, pulses, channel)
- def **SetOpenLoopMoveBack** (self, secondary, pulses, channel)
- def **SetDisabled** (self, secondary, disable)
- def **SetCurrentStatusInExternalTrigger** (self, secondary, triggerMode)
- def **SetPositionLimit** (self, secondary, minvalue, maxvalue)
- def **SetJoystickConfig** (self, secondary, value, value2)

- def **SetStepPulseAndResponse** (self, secondary, value)

Static Public Member Functions

- def **ListDevices** ()
- def **load_library** (path)

Data Fields

- **hdl**

Static Public Attributes

- **pdxcLib** = None
- bool **isLoad** = False

3.1.2. Function Documentation

Constructor & Destructor Documentation

- def ***__init__*** (self)

Member Function Documentation

- def ***Close*** (self)

```
Close opened device
Returns:
    0: Success; negative number: failed.
```

- def ***GetAbnormalMoveDetect*** (*self*, *secondary*, *enable*)

```
Get whether abnormal move detect enable
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    enable: ('1' : enabled, '0' : disabled)
Returns:
    0: Success; negative number: failed.
```

- def ***GetAllParametersInExternalTrigger*** (*self*, *secondary*, *alltriggerState*)

```
Get all parameters except for Manual mode which has no parameters, they are
"AR/AF,FR/FF[value],PR/PF[pos1],PR/PF[pos2]", which stands for Analog - In mode with rising / falling
edge, Fixed - Size mode with rising / falling edge and value defined in PDX1(mm)/ PDX1(mm)/PDXR(°)
unit, and Two - Position - Switching mode with rising / fallingfor each position, assigned by pos1
and pos2.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    alltriggerState: device all trigger State
Returns:
    0: Success; negative number: failed.
```

- def ***GetAnalogInputGain*** (*self*, *secondary*, *aiGain*)

```
Get gain value of analog input
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
```

```
    aiGain: analog input gain,the range is: PDX1/PDX2:[0.1,1]; PDXR1:[0.1,100]
Returns:
    0: Success; negative number: failed.
```

- def ***GetAnalogInputOffSet*** (*self*, *secondary*, *aiOffSet*)

```
Get OffSet value of analog input
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    aiOffSet: analog input offset: PDX1:[-10,10]mm; PDX2:[-2.5, 2.5]mm; PDXR:[-999.9,999.9]°
Returns:
    0: Success; negative number: failed.
```

- def ***GetAnalogOutGain*** (*self*, *secondary*, *aoGain*)

```
Get gain value of analog output
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    aoGain: analog output gain,the range is [0.1,1]V
Returns:
    0: Success; negative number: failed.
```

- def ***GetAnalogOutOffSet*** (*self*, *secondary*, *aoOffSet*)

```
Get OffSet value of analog output
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    aoOffSet: analog output offset:PDX1/PDX2:[-10,10]mm; PDXR:[-999.9,999.9]°
Returns:
    0: Success; negative number: failed.
```

- def ***GetBackwardAmplitude*** (*self*, *secondary*, *baampli*)

```
Get the backward amplitude for PD2/PD3/SMC
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    baampli: backward amplitude (10~100)
Returns:
    0: Success; negative number: failed.
```

- def ***GetCalibrationIsCompleted*** (*self*, *secondary*, *homed*)

```
Get 'YES' when calibration complete, 'NO' when not
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    homed: device calibration completed status
Returns:
    0: Success; negative number: failed.
```

- def ***GetCurrentPosition*** (*self*, *secondary*, *position*)

```
Get current position counter.It only returns position value in D - SUB mode, other will return
warnings
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    position: current position: PDX1[-10,10]mm; PDX2[-2.5,2.5]mm; PDXR:[-999.9,999.9] °
Returns:
    0: Success; negative number: failed.
```

- def ***GetCurrentStatus*** (*self*, *secondary*, *status*)

```
In D - sub Mode, it will return strings consist of ERR, KP, KI, KD, Current Loop, Velocity Level(open
loop), Step size(open loop), Speed(Closed loop), Jog Step(closed loop), Home Flag and Abnormal
Detection Status, parameters in one command, results will be separated by ',' for each segment. In
SMC Mode, it will return strings consist of ERR, Velocity and Step Size for Channel 1, Velocity and
Step Size for channel 2.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    status: device status
Returns:
    0: Success; negative number: failed.
```

- def ***GetCurrentStatusInExternalTrigger*** (*self*, *secondary*, *triggerState*)

```
Get the current status of external trigger mode, they are "ML"(Manual mode), "AR/AF"(Analog-In mode
with rising/falling edge), "FR/FF[value]"(Fixed-Step size mode), and "PR/PF[pos1],PR/PF[pos2]"(Two-
Postion Switching mode).
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    triggerState: status of external trigger mode
Returns:
    0: Success; negative number: failed.
```

- def ***GetDaisyChainStatus*** (*self*, *secondary*, *status*)

```
Get the current status, these are "Single-Mode", "Chain-Mode Main", "Chain-Mode Secondary1", "Chain-
Mode Secondary2", .. "Chain-Mode Secondary8". Daisy - chain query command format is as following,
M0:<CMD ? >_Sx : <CMD ? >_Sy : <CMD ? ><CR> M0 means the fixed Main, while Sx means x - th Secondary,
x starts from 1 to 11. <CMD ? > means listed Query, eg.POS ?, and the return code is begin with "Sx:".
Args:
    secondary: index in daisy chain (0:Single Mode, 1:Main, 2 -12 : Secondary1 - Secondary11)
    status: current Daisy - chain status
Returns:
    0: Success; negative number: failed.
```

- def ***GetDisabled*** (*self*, *secondary*, *disable*)

```
Get device disable state, under disable state stage will not move.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    disable: disable state( 0 :enabled, 1 :disabled)
Returns:
    0: Success; negative number: failed.
```

- def ***GetErrorMessage*** (*self*, *secondary*, *error*)

```
Get error message code
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    error: error code
   0 : None errors occurred,
   1 : command not defined,
   2 : command Data out - of - range,
   3 : device failed to execute last command,
   4 : no waveform data loaded,
   5 : Stage need Home first,
   6 : device works in the wrong mode,
   7 : stage move abnormal,
   8  : excessive current occurred,
   9 : over temperature occurred,
  17 : unkown error occurred)
Returns:
    0: Success; negative number: failed.
```

- def ***GetForwardAmplitude*** (*self*, *secondary*, *forampli*)

```
Get the forward amplitude for PD2/PD3/SMC
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    forampli: forward amplitude (10~100)
Returns:
    0: Success; negative number: failed.
```

- def ***GetFV*** (*self*, *secondary*, *fv*)

```
Get strings, with former one is firmware version, the latter one is hardware version, the two are
seperated by comma ','
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    fv: firmware version and hardware version
Returns:
    0: Success; negative number: failed.
```

- def ***GetHandle*** (*self*, *serialNo*)

```
get handle of port
Args:
    serialNo: serial number of the device to be checked.
Returns:
    0: -1:no handle  non-negtive number: handle..
```

- def ***GetJoystickConfig*** (*self*, *secondary*, *value*)

```
Get the config of joystick
Args:
    secondary: index in daisy chain (0:Single Mode or Main)
    value: Return the array of X,Y,x=0 means no device, x=1 means Single-Mode or Main, x=2 to 12
means
   Secondary1 to Secondary11, y means the number of knob on joystick(from 1 to n),each group of data
is
   separated by space such as "[1,1  2,2  4,3]".
Returns:
    0: Success; negative number: failed.
```

- def ***GetJoystickStatus*** (*self*, *secondary*, *number*)

```
Get the joystick status.
Args:
    secondary: index in daisy chain (0:Single Mode or Main)
    number: The number of knob on joystick. 0: No joystick is connected to the device
Returns:
    0: Success; negative number: failed.
```

- def ***GetKdOfPidParameters*** (*self*, *secondary*, *Kd*)

```
Get current Kd value
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    Kd: Kd of PID
Returns:
    0: Success; negative number: failed.
```

- def ***GetKiOfPidParameters*** (*self*, *secondary*, *Ki*)

```
Get current Ki value
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    Ki: Ki of PID
Returns:
```

```
    0: Success; negative number: failed.
```

- def ***GetKpOfPidParameters*** (*self*, *secondary*, *Kp*)

```
Get current Kp value
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    Kp: Kp of PID
Returns:
    0: Success; negative number: failed.
```

- def ***GetLoopStatus*** (*self*, *secondary*, *loop*)

```
Get device loop status
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    loop: loop status(1 :open loop, 0 :close loop)
Returns:
    0: Success; negative number: failed.
```

- def ***GetOpenLoopFrequency*** (*self*, *secondary*, *fry*)

```
Get current frequency of open loop of channel 1 in SMC mode.The unit is Hz
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    fry: current frequency of open loop
Returns:
    0: Success; negative number: failed.
```

- def ***GetOpenLoopFrequency2*** (*self*, *secondary*, *fry2*)

```
Get current frequency of open loop of channel 2 in SMC mode.The unit is Hz
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    fry2: current frequency of open loop
Returns:
    0: Success; negative number: failed.
```

- def ***GetOpenLoopFrequency3*** (*self*, *secondary*, *fry3*)

```
Get current frequency of open loop for PD2/PD3.The unit is Hz.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    fry3: current frequency of open loop
Returns:
    0: Success; negative number: failed.
```

- def ***GetOpenLoopJogSize*** (*self*, *secondary*, *stepsize*)

```
Get the step size for open - loop for Channel 1
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    stepsize: open loop jog step size (1~65535)
Returns:
    0: Success; negative number: failed.
```

- def ***GetOpenLoopJogSize2*** (*self*, *secondary*, *stepsize2*)

```
Get the step size for open - loop for Channel 2
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    stepsize2: open loop jog step size (1~65535)
```

```
Returns:
    0: Success; negative number: failed.
```

- def ***GetOpenLoopJogSize3*** (*self*, *secondary*, *stepsize3*)

```
Get the step size for open - loop for PD2/PD3
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    stepsize3: open loop jog step size (1~65535)
Returns:
    0: Success; negative number: failed.
```

- def ***GetPositionLimit*** (*self*, *secondary*, *limit*)

```
Get values of positon limit by format of [Min, Max].
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    limit: Positon limit. Format of [Min, Max]
Returns:
    0: Success; negative number: failed.
```

- def ***GetSN*** (*self*, *secondary*, *SN*)

```
Get the SN of PDXC controller.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    SN: PDXC controller SN
Returns:
    0: Success; negative number: failed.
```

- def ***GetSN2*** (*self*, *secondary*, *SN2*)

```
Get the Serial-Number followed by Item-Number of stage connected to PDXC controller, with the former
one consist of strictly 8 bytes, and the latter one is not definte(e.g. when we get "SN024680PDX1/M",
"SN024680" represent SN, "PDX1/M" represent Item-Number.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    SN2: Serial-Number and Item-Number of stage
Returns:
    0: Success; negative number: failed.
```

- def ***GetSpeedStageType*** (*self*, *secondary*, *type*)

```
Get the current type of stage
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    type: stage type(0 :PDX1/PDX2/PDXR stage, 1 :SMC stage, 2 :PD2/PD3 stage)
Returns:
    0: Success; negative number: failed.
```

- def ***GetTargetTriggerPosition*** (*self*, *secondary*, *position*)

```
Get the target position which is calculated based on Analog In Gain and Analog In Offset.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    position: target position: PDX1[-10,10]; PDX2[-2.5,2.5]; PDXR:[-999.9,999.9]
Returns:
    0: Success; negative number: failed.
```

- def ***GetUserDataIsSaved*** (*self*, *secondary*, *saved*)

```
Get data saved status
```

```
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    saved: saved status(bit 0 : close - loop stage, bit 1 : SMC stages [0 : no user data at all,
   1 : only close - loop stage , 2 : only SMC stages , 3 : both close - loop and SMC stages.])
Returns:
    0: Success; negative number: failed.
```

- def ***IsOpen*** (*self*, *serialNo*)

```
Check opened status of device
Args:
    serialNo: serial number of device
Returns:
    0: device is not opened; 1: device is opened.
```

- def ***ListDevices*** ()`[static]`

```
List all connected pdxc devices
Returns:
   The pdxc device list, each deice item is serialNumber/COM
```

- def load_library (*path*)`[static]`

- def ***Open*** (*self*, *serialNo*, *nBaud*, *timeout*)

```
Open device
Args:
    serialNo: serial number of pdxc device
    nBaud: bit per second of port
    timeout: set timeout value in (s)
Returns:
    non-negative number: hdl number returned Successful; negative number: failed.
```

- def ***SetAbnormalMoveDetect*** (*self*, *secondary*, *enable*)

```
Switch on or off the abnormal move detection, default is on.Used to detect stage stuck or move by
external force.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    enable: detect enable(1 : Enable , 0 :Disable)
Returns:
    0: Success; negative number: failed.
```

- def ***SetAllCustomerData*** (*self*, *secondary*, *saveState*)

```
Data contains SMC data and non - SMC data, the former one contains velocity and step size for each
channel,and the later one contains status(reserved), speed for closedloop, Velocity Level for penloop,
position and step size, daisy - chain number, input trigger value abnormal detection value input /
output gain and offset(4 in all), and step distance.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    saveState: set to save or erase(1:save data;0:erase data.)
Returns:
    0: Success; negative number: failed.
```

- def ***SetAnalogInputGain*** (*self*, *secondary*, *aiGain*)

```
Change gain value of analog input, default input voltage range is[-10V, 10V], standing for PDX1[+ -
10mm]/ PDX2[+ -2.5mm]/PDXR[+ - 10°] position range.If change value of Gain&Offset, we will get new
input voltage range.The analog input gain vaule(ING), calculated by(max - min) / 20, analog input
offSet value(INO) calculated by(min + max) / 2. E.g.If new wanted input range is in[0V, 5V], which
means analog input gain is 0.25, analog input offSet value is 2.5. (the new Voltage Sampled is
```

```
function of Vold, ING and INO Vnew = ING * Vold + 1.5(1 - ING) - 0.15*INO, where Vout is default
range's output.)
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    aiGain: gain value of analog input,the range is: PDX1/PDX2:[0.1,1]; PDXR1:[0.1,100]
Returns:
    0: Success; negative number: failed.
```

- def ***SetAnalogInputOffSet*** (*self*, *secondary*, *aiOffset*)

```
Change offset value of analog input, default input voltage range is[-10V, 10V].If change value of
Gain&Offset, we will get new input voltage range.The analog input gain vaule(ING), calculated by(max
- min) / 20, analog input offSet value(INO) calculated by(min + max)/ 2. E.g. If new wanted input
range is in[0V, 5V], which means analog input gain is 0.25, analog input offSet value is 2.5.(the new
Voltage Sampled is function of Vold, ING and INO Vnew = ING * Vold + 1.5(1 - ING) - 0.15*INO, where
Vout is default range's output.)
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    aiOffset: offset value of analog input:PDX1:[-10,10]mm; PDX2:[-2.5, 2.5]mm; PDXR:[-999.9,999.9]°
Returns:
    0: Success; negative number: failed.
```

- def ***SetAnalogOutGain*** (*self*, *secondary*, *aoGain*)

```
Change gain Value of analog output, default output range is[-10V, 10V].If change value of Gain&Offset,
we will get new output range.The analog out gain value(OUG), The analog out offset value(OUO).E.g.If
new wanted output range is in[0V, 5V], which means OUG = 0.25, OUO = 2.5. (the new voltage output for
DAC is function of position, which equals Vdac_new = (Vdac_old - 1.5)*OUG + OUO / 6.8 + 1.5, where
Vdac_old = (pos / 100000 + 10)/ 20 * 3).
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    aoGain: gain Value of analog output, the range is [0.1,1]V
Returns:
    0: Success; negative number: failed.
```

- def ***SetAnalogOutOffSet*** (*self*, *secondary*, *aoOffset*)

```
Change offset Value of analog output:[-10,10].If change value of Gain&Offset, wewill get new output
range.The analog out gain value(OUG),The analog out offset value(OUO).E.g.If new wanted output range
is in[0V, 5V], which means OUG = 0.25, OUO = 2.5. (the new voltage output for DAC is function of
position, which equals Vdac_new =(Vdac_old - 1.5)*OUG + OUO / 6.8 + 1.5, where Vdac_old = (pos /
100000 + 10) / 20 * 3).
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    aoOffset: offset Value of analog output: PDX1/PDX2:[-10,10]mm; PDXR:[-999.9,999.9]°
Returns:
    0: Success; negative number: failed.
```

- def ***SetBackwardAmplitude*** (*self*, *secondary*, *baampli*)

```
Set the forward amplitude for PD2/PD3/SMC
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    baampli: forward amplitude (10~100)
Returns:
    0: Success; negative number: failed.
```

- def ***SetCurrentStatusInExternalTrigger*** (*self*, *secondary*, *triggerMode*)

```
Set the current status of external trigger mode
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    triggerMode: "ML"(Manual mode), "AR/AF"(Analog - In mode with rising / falling edge),
"FR/FF[value]"(Fixed
    - Step size mode), and "PR/PF[pos1],PR/PF[pos2]"(Two - Postion Switching mode).
```

```
Returns:
    0: Success; negative number: failed.
```

- def ***SetDaisyChain*** (*self*, *index*)

```
Set device position in daisy-chain.
Args:
    index: index in daisy chain (0:Single Mode, 1:Main, 2 -12 : Secondary1 - Secondary11)
Returns:
    0: Success; negative number: failed.
```

- def ***SetDisabled*** (*self*, *secondary*, *disable*)

```
Set controller disabled
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    disable: disable state(0:enable the device, 1:disable the device)
Returns:
    0: Success; negative number: failed.
```

- def ***SetForwardAmplitude*** (*self*, *secondary*, *forampli*)

```
Set the forward amplitude for PD2/SMC
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    forampli: forward amplitude (10~100)
Returns:
    0: Success; negative number: failed.
```

- def ***SetJoystickConfig*** (*self*, *secondary*, *value*, *value2*)

```
Set the joystick config for device.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    value: 0:No device; 1:Single-Mode or Main; 2 -12 : Secondary1-Secondary11
    value2: The number of knob on joystick[from 0 to n(0:no knob; 1:fixed to 1:Single-Mode or Main)]
Returns:
    0: Success; negative number: failed.
```

- def ***SetKdOfPidParameters*** (*self*, *secondary*, *Kd*)

```
Set current Kd value, will store in Flash memory.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    Kd: Kd of PID
Returns:
    0: Success; negative number: failed.
```

- def ***SetKiOfPidParameters*** (*self*, *secondary*, *Ki*)

```
Set current Ki value, will store in Flash memory.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    Ki: Ki of PID
Returns:
    0: Success; negative number: failed.
```

- def ***SetKpOfPidParameters*** (*self*, *secondary*, *Kp*)

```
Set current Kp value, will store in Flash memory.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
```

```
    Kp: Kp of PID
Returns:
    0: Success; negative number: failed.
```

- def ***SetLoop*** (*self*, *secondary*, *loop*)

```
Switch closeloop and open loop
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    loop: loop type (1 : open loop (default), 0 : close loop.)
Returns:
    0: Success; negative number: failed.
```

- def ***SetOpenLoopFrequency*** (*self*, *secondary*, *fry*)

```
Set the open loop frequency of channel 1.The unit is Hz. Must in SMC mode
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    fry: current frequency of open loop
Returns:
    0: Success; negative number: failed.
```

- def ***SetOpenLoopFrequency2*** (*self*, *secondary*, *fry2*)

```
Set the open loop frequency of channel 2.The unit is Hz. Must in SMC mode
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    fry2: current frequency of open loop
Returns:
    0: Success; negative number: failed.
```

- def ***SetOpenLoopFrequency3*** (*self*, *secondary*, *fry3*)

```
Set the output frequency of D-Sub stage without encoder(PD2).The unit is Hz.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    fry3: current frequency of open loop
Returns:
    0: Success; negative number: failed.
```

- def ***SetOpenLoopJogSize*** (*self*, *secondary*, *stepsize*)

```
Set the step size for open - loop for Channel 1
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    stepsize: open loop jog step size (1~65535)
Returns:
    0: Success; negative number: failed.
```

- def ***SetOpenLoopJogSize2*** (*self*, *secondary*, *stepsize2*)

```
Set the step size for open - loop for Channel 2
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    stepsize2: open loop jog step size (1~65535)
Returns:
    0: Success; negative number: failed.
```

- def ***SetOpenLoopJogSize3*** (*self*, *secondary*, *stepsize3*)

```
Set the step size for open - loop for PD2/PD3
Args:
```

```
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    stepsize3: open loop jog step size (1~400000)
Returns:
    0: Success; negative number: failed.
```

- def ***SetOpenLoopMoveBack*** (*self*, *secondary*, *pulses*, *channel*)

```
Set Open Loop Move Forward
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    pulses: pulses of move channel:SMC[1,65535]; PD2/PD3[1,400000]
    channel: Move fordward channel (0 : channel 1, 1 : channel 2, others :both channels)
Returns:
    0: Success; negative number: failed.
```

- def ***SetOpenLoopMoveForward*** (*self*, *secondary*, *pulses*, *channel*)

```
Set Open Loop Move Forward
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    pulses: pulses of move channel:SMC[1,65535]; PD2/PD3[1,400000]
    channel: Move fordward channel (0 : channel 1, 1 : channel 2, others :both channels)
Returns:
    0: Success; negative number: failed.
```

- def ***SetPositionCalibration*** (*self*, *secondary*, *home*)

```
Calibrate the QDEC counter after power - up.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    home: controller home(1: Yes, 0:No)
Returns:
    0: Success; negative number: failed.
```

- def ***SetPositionLimit*** (*self*, *secondary*, *minvalue*, *maxvalue*)

```
Set the minimum and maximum values of position, the controller will not respond to the value surpass
this range.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    minvalue: Minimum value of position, PDX1 range[-10,max] mm; PDX2 range[-2.5,max] mm; PDXR
range[-999.9,max] °
    maxvalue: Maximum value of position, PDX1 range[min,10] mm; PDX2 range[min,2.5] mm; PDXR
range[min.9,999.9] °
Returns:
    0: Success; negative number: failed.
```

- def ***SetStepPulseAndResponse*** (*self*, *secondary*, *value*)

```
Set step position
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    value: relative distance to be moved: PDX1:[-10,-0.00001]mm,[0.00001,10]mm; PDX2: [-2.5, -
0.0003]mm, [0.0003, 2.5]mm; PDXR:[-180,-0.00005] °, [0.00005,180] °
Returns:
    0: Success; negative number: failed.
```

- def ***SetTargetPosition*** (*self*, *secondary*, *position*)

```
Set the target position counter.Only work in Manual Mode. In Open Loop, will use continuous pulses at
preset amplitude until reach the destination without PID control(including Speed and Acurate osition).
Work in both Open and Closed Loop in D - SUB mode only. While in Closed Loop, will add PID control,
and anlaog move near destination.
```

```
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    position: Target Position: PDX1[-10,10]mm; PDX2[-2.5,2.5]mm; PDXR:[-180,180]°
Returns:
    0: Success; negative number: failed.
```

- def ***SetTargetSpeed*** (*self*, *secondary*, *speed*)

```
Set the desired speed. Work only under D- SUB mode, other will return warnings.
Args:
    secondary: index in daisy chain (0:Single Mode or Main, 1 -11 : Secondary1 - Secondary11)
    speed: desired speed(PDX1:2-20 mm/s; PDX2:1-10 mm/s; PDXR:10-30 °/s)
Returns:
    0: Success; negative number: failed.
```

Field Documentation

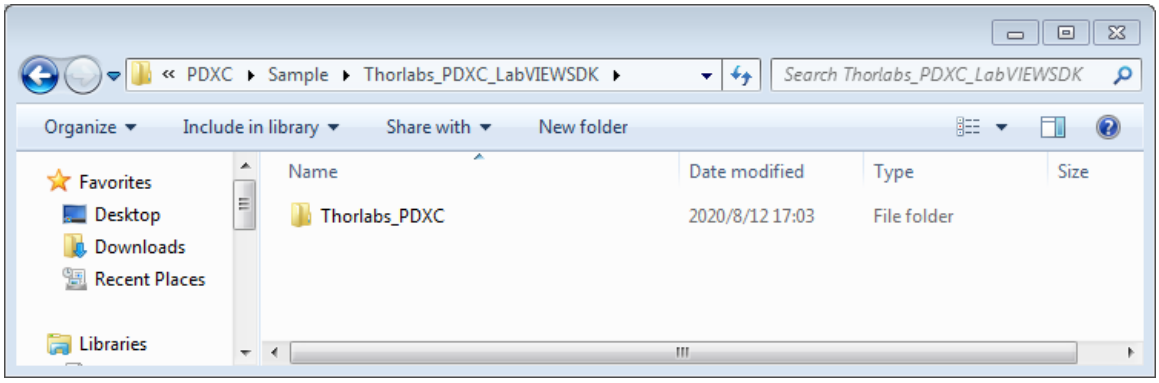
- ***hdl***
- ***bool* isLoad** = False `[static]`
- ***pdxcLib*** = None `[static]`

Chapter 4 LabVIEW Software Development Kit

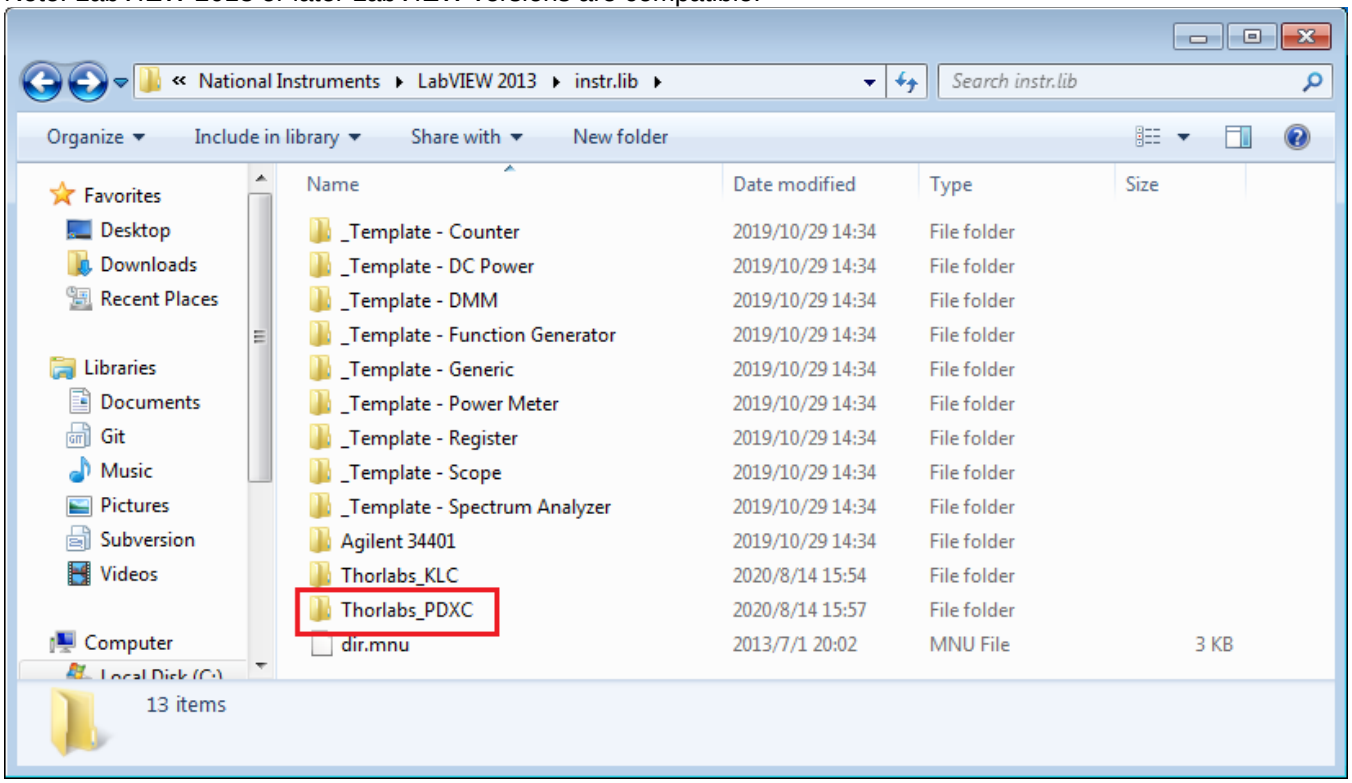
The user can start software development with LabVIEW 2013 or later versions based on LabVIEW instrument driver mechanism. The supported files are in **\Thorlabs_PDXC_LabVIEWSDK** under the **Sample** directory.

How to install

Copy to instr.lib folder under LabVIEW installation folder.

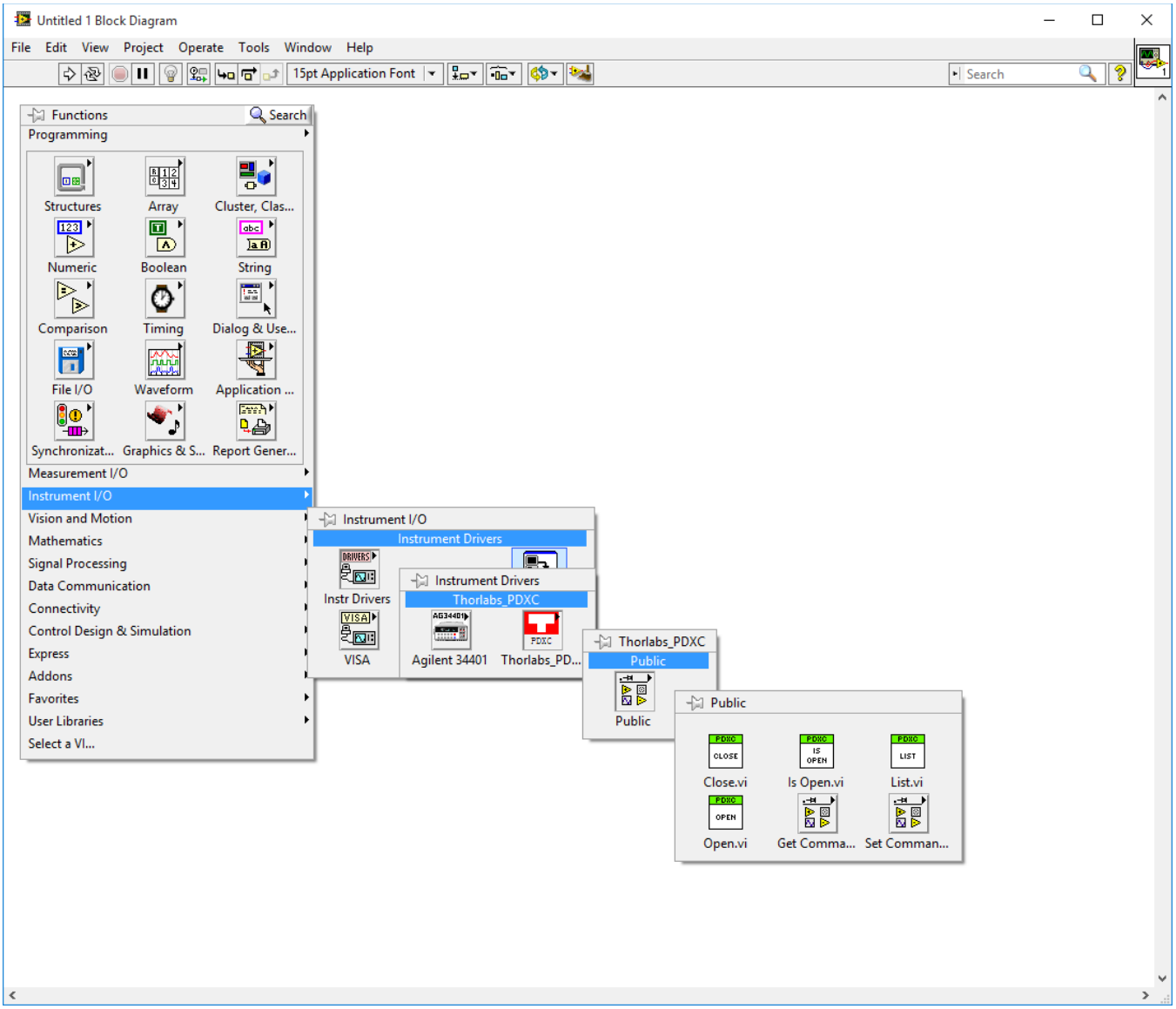


Destination folder: under %LabVIEW install path%\instr.lib
Typically, C:\Program Files (x86)\National Instruments\LabVIEW 2013\instr.lib
Note: LabVIEW 2013 or later LabVIEW versions are compatible.



How to find VI

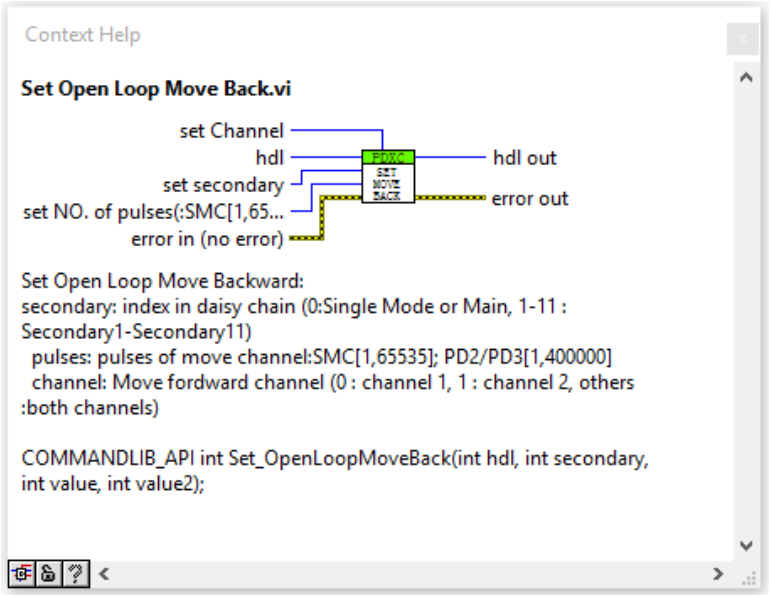
VI Could be found under: Functions\Instrument I/O\Instrument Drivers\



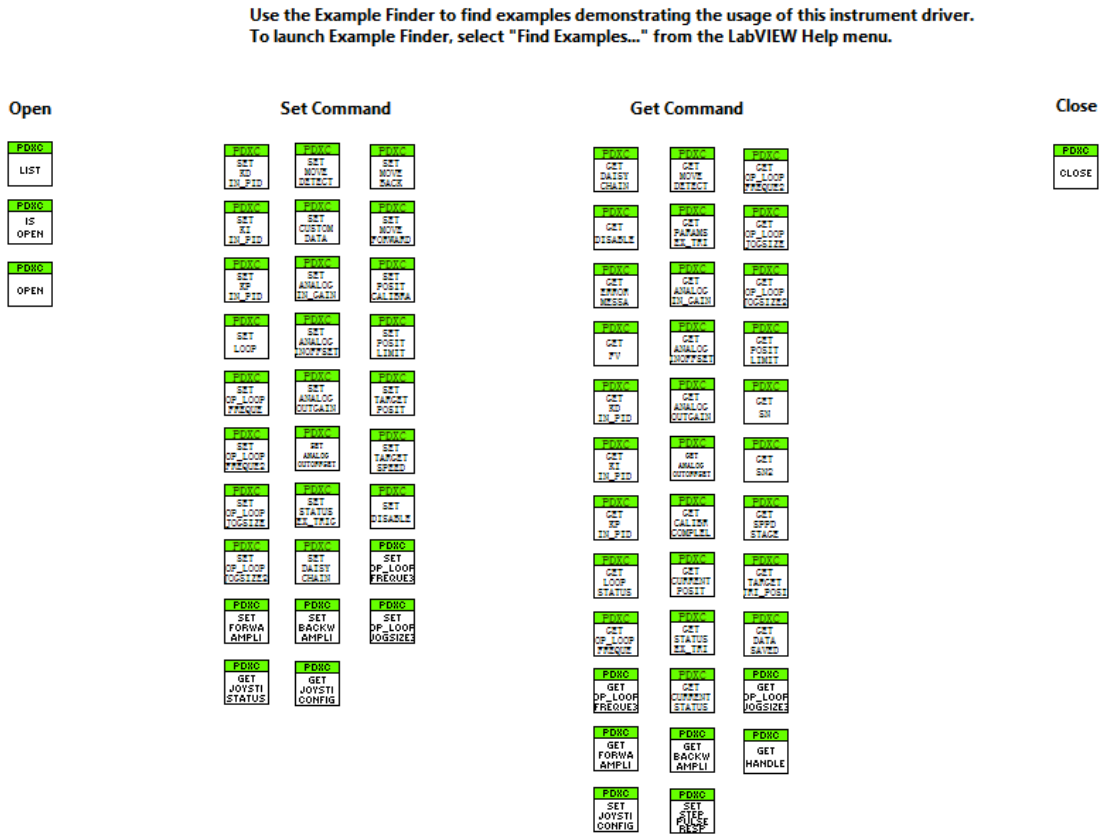
How to use

1. From VI

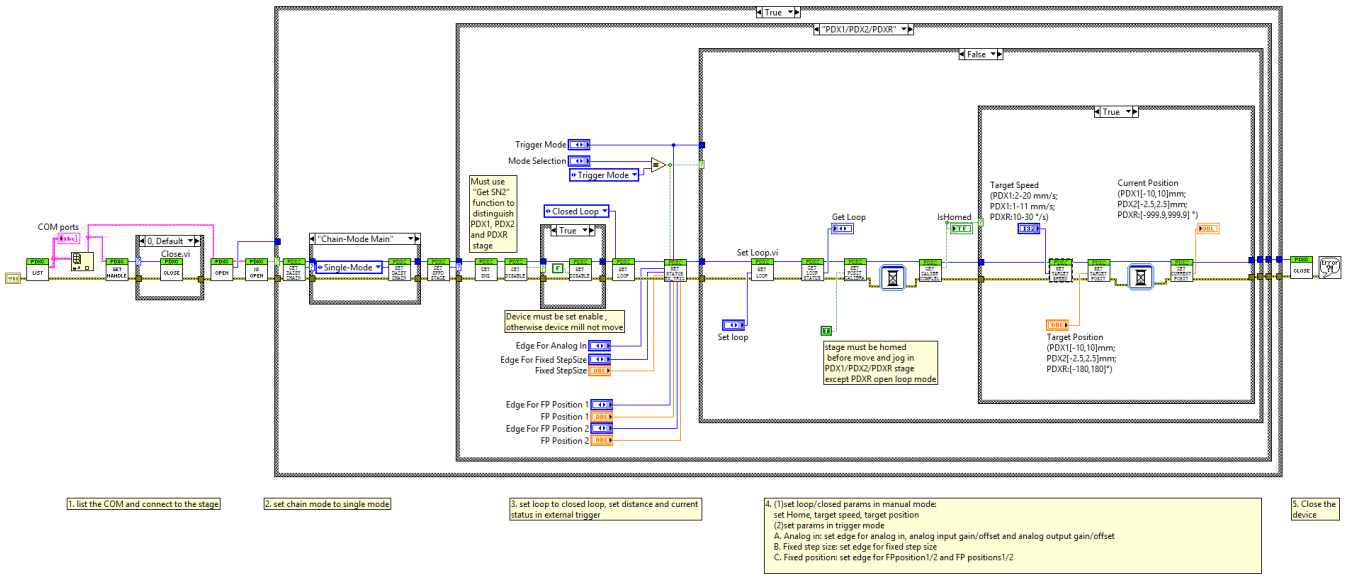
Note: Before you open the SDK LabVIEW project, make sure the device has been connected to the computer.



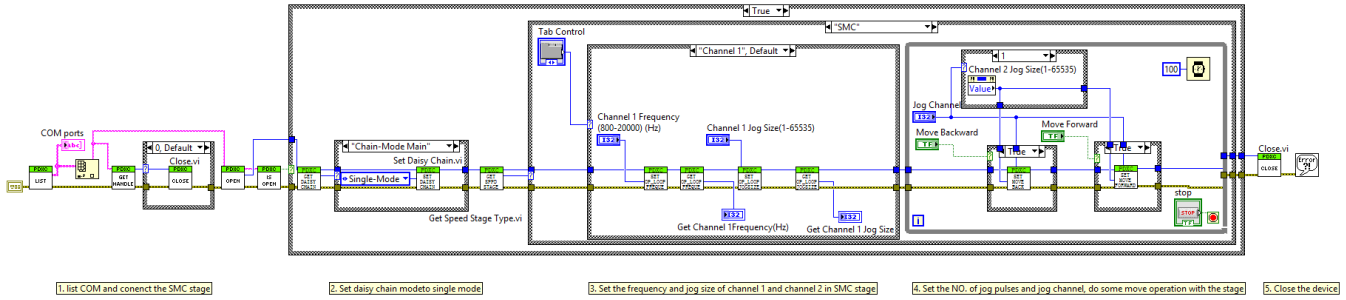
2. From VI tree
Some classic data flow in VI tree.



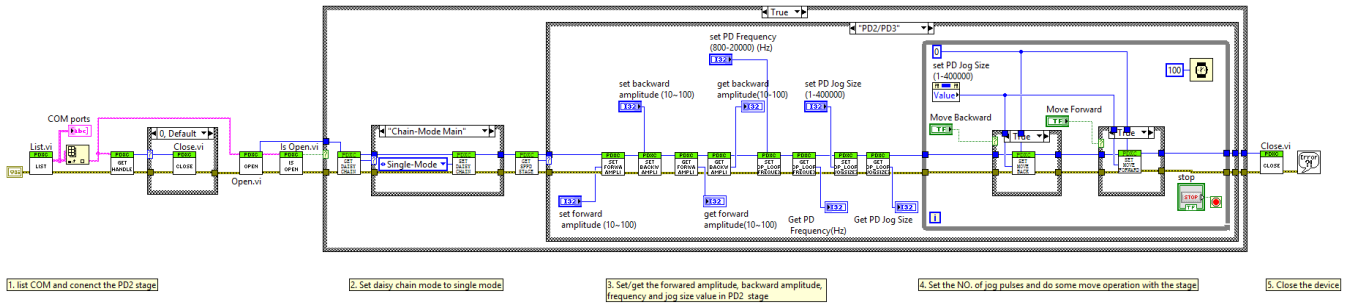
3. From example
Examples show the classic ***single mode*** usage. Examples' path: instr.lib\Thorlabs_PDXC\Example
(1) D-sub PDX1/PDX2/PDXR example



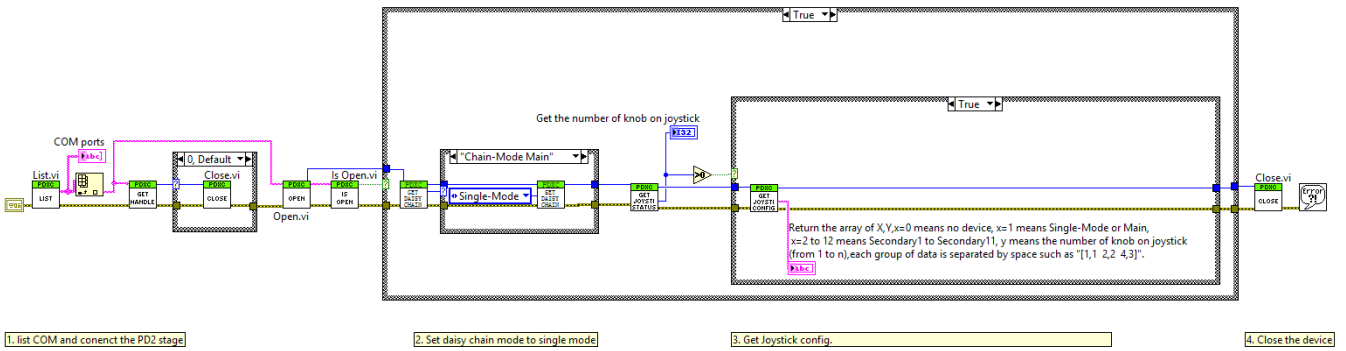
(2) SMC example



(3) D-sub PD2/PD3 example



(4) Joystick example




UNLEASHED