

libximc

3.0.2

Generated on Tue Dec 16 2025 15:52:24 for libximc by Doxygen 1.14.0

Tue Dec 16 2025 15:52:24

1 libximc library	1
1.1 What the controller does	1
1.2 What can do libximc library	1
1.3 Assistance	2
2 Introduction	3
2.1 About library	3
2.1.1 Supported OS and environment requirements:	3
3 How to use with...	4
3.0.1 Visual C++	4
3.0.2 CodeBlocks	4
3.0.3 MinGW	5
3.0.4 C++ Builder	5
3.0.5 XCode	5
3.0.6 GCC	5
3.0.7 .NET	6
3.0.8 Java	6
3.0.9 Python	6
3.0.10 MATLAB	7
3.1 Generic logging facility	7
3.2 Required permissions	7
3.3 C-profiles	7
3.4 Python-profiles	7
4 Working with user units	8
4.1 The structure of the conversion units calibration_t	8
4.2 Alternative functions for working with user units and data structures for them	8
4.3 Coordinate correction table for more accurate positioning	9
5 Data Structure Documentation	10
5.1 accessories_settings_t Struct Reference	10
5.1.1 Detailed Description	11
5.1.2 Field Documentation	11
5.1.2.1 LimitSwitchesSettings	11
5.1.2.2 MagneticBrakeInfo	11
5.1.2.3 MBRatedCurrent	11
5.1.2.4 MBRatedVoltage	11
5.1.2.5 MBSettings	11
5.1.2.6 MBTorque	12
5.1.2.7 TemperatureSensorInfo	12
5.1.2.8 TSGrad	12

5.1.2.9 TSMaX	12
5.1.2.10 TSMin	12
5.1.2.11 TSSettings	12
5.2 analog_data_t Struct Reference	12
5.2.1 Detailed Description	14
5.2.2 Field Documentation	14
5.2.2.1 A1Voltage	14
5.2.2.2 A1Voltage_ADC	14
5.2.2.3 A2Voltage	14
5.2.2.4 A2Voltage_ADC	14
5.2.2.5 ACurrent	15
5.2.2.6 ACurrent_ADC	15
5.2.2.7 B1Voltage	15
5.2.2.8 B1Voltage_ADC	15
5.2.2.9 B2Voltage	15
5.2.2.10 B2Voltage_ADC	15
5.2.2.11 BCurrent	15
5.2.2.12 BCurrent_ADC	16
5.2.2.13 Enc_Check	16
5.2.2.14 Enc_Check_ADC	16
5.2.2.15 FullCurrent	16
5.2.2.16 FullCurrent_ADC	16
5.2.2.17 Joy	16
5.2.2.18 Joy_ADC	16
5.2.2.19 L	17
5.2.2.20 Pot	17
5.2.2.21 R	17
5.2.2.22 SupVoltage	17
5.2.2.23 SupVoltage_ADC	17
5.2.2.24 Temp	17
5.2.2.25 Temp_ADC	17
5.3 brake_settings_t Struct Reference	18
5.3.1 Detailed Description	18
5.3.2 Field Documentation	18
5.3.2.1 BrakeFlags	18
5.3.2.2 t1	18
5.3.2.3 t2	19
5.3.2.4 t3	19
5.3.2.5 t4	19
5.4 calibration_settings_t Struct Reference	19
5.4.1 Detailed Description	20

5.4.2 Field Documentation	20
5.4.2.1 CSS1_A	20
5.4.2.2 CSS1_B	20
5.4.2.3 CSS2_A	20
5.4.2.4 CSS2_B	20
5.4.2.5 FullCurrent_A	20
5.4.2.6 FullCurrent_B	21
5.5 calibration_t Struct Reference	21
5.5.1 Detailed Description	21
5.5.2 Field Documentation	22
5.5.2.1 A	22
5.6 chart_data_t Struct Reference	22
5.6.1 Detailed Description	23
5.6.2 Field Documentation	23
5.6.2.1 AveragedPowerRatio	23
5.6.2.2 Joy	23
5.6.2.3 Pot	23
5.6.2.4 WindingCurrentA	23
5.6.2.5 WindingCurrentB	23
5.6.2.6 WindingCurrentC	24
5.6.2.7 WindingVoltageA	24
5.6.2.8 WindingVoltageB	24
5.6.2.9 WindingVoltageC	24
5.7 control_settings_calb_t Struct Reference	24
5.7.1 Detailed Description	25
5.7.2 Field Documentation	25
5.7.2.1 Flags	25
5.7.2.2 MaxClickTime	25
5.7.2.3 MaxSpeed	25
5.7.2.4 Timeout	25
5.8 control_settings_t Struct Reference	26
5.8.1 Detailed Description	26
5.8.2 Field Documentation	26
5.8.2.1 Flags	26
5.8.2.2 MaxClickTime	27
5.8.2.3 MaxSpeed	27
5.8.2.4 Timeout	27
5.8.2.5 uDeltaPosition	27
5.8.2.6 uMaxSpeed	27
5.9 controller_name_t Struct Reference	27
5.9.1 Detailed Description	28

5.9.2 Field Documentation	28
5.9.2.1 ControllerName	28
5.9.2.2 CtrlFlags	28
5.10 ctp_settings_t Struct Reference	28
5.10.1 Detailed Description	29
5.10.2 Field Documentation	29
5.10.2.1 CTPFlags	29
5.10.2.2 CTPMinError	29
5.11 debug_read_t Struct Reference	29
5.11.1 Detailed Description	30
5.11.2 Field Documentation	30
5.11.2.1 DebugData	30
5.12 debug_write_t Struct Reference	30
5.12.1 Detailed Description	30
5.12.2 Field Documentation	31
5.12.2.1 DebugData	31
5.13 device_information_t Struct Reference	31
5.13.1 Detailed Description	31
5.13.2 Field Documentation	31
5.13.2.1 Major	31
5.13.2.2 Minor	32
5.13.2.3 Release	32
5.14 device_network_information_t Struct Reference	32
5.14.1 Detailed Description	32
5.15 edges_settings_calb_t Struct Reference	32
5.15.1 Detailed Description	33
5.15.2 Field Documentation	33
5.15.2.1 BorderFlags	33
5.15.2.2 EnderFlags	33
5.15.2.3 LeftBorder	33
5.15.2.4 RightBorder	34
5.16 edges_settings_t Struct Reference	34
5.16.1 Detailed Description	34
5.16.2 Field Documentation	35
5.16.2.1 BorderFlags	35
5.16.2.2 EnderFlags	35
5.16.2.3 LeftBorder	35
5.16.2.4 RightBorder	35
5.16.2.5 uLeftBorder	35
5.16.2.6 uRightBorder	35
5.17 emf_settings_t Struct Reference	36

5.17.1 Detailed Description	36
5.17.2 Field Documentation	36
5.17.2.1 BackEMFFlags	36
5.17.2.2 Km	36
5.17.2.3 L	37
5.17.2.4 R	37
5.18 encoder_information_t Struct Reference	37
5.18.1 Detailed Description	37
5.18.2 Field Documentation	37
5.18.2.1 Manufacturer	37
5.18.2.2 PartNumber	38
5.19 encoder_settings_t Struct Reference	38
5.19.1 Detailed Description	38
5.19.2 Field Documentation	38
5.19.2.1 EncoderSettings	38
5.19.2.2 MaxCurrentConsumption	39
5.19.2.3 MaxOperatingFrequency	39
5.19.2.4 SupplyVoltageMax	39
5.19.2.5 SupplyVoltageMin	39
5.20 engine_advanced_setup_t Struct Reference	39
5.20.1 Detailed Description	40
5.20.2 Field Documentation	40
5.20.2.1 stepcloseloop_Kp_high	40
5.20.2.2 stepcloseloop_Kp_low	40
5.20.2.3 stepcloseloop_Kw	40
5.21 engine_settings_calb_t Struct Reference	40
5.21.1 Detailed Description	41
5.21.2 Field Documentation	41
5.21.2.1 Antiplay	41
5.21.2.2 EngineFlags	41
5.21.2.3 MicrostepMode	42
5.21.2.4 NomCurrent	42
5.21.2.5 NomSpeed	42
5.21.2.6 NomVoltage	42
5.21.2.7 StepsPerRev	42
5.22 engine_settings_t Struct Reference	42
5.22.1 Detailed Description	43
5.22.2 Field Documentation	43
5.22.2.1 Antiplay	43
5.22.2.2 EngineFlags	44
5.22.2.3 MicrostepMode	44

5.22.2.4	NomCurrent	44
5.22.2.5	NomSpeed	44
5.22.2.6	NomVoltage	44
5.22.2.7	StepsPerRev	44
5.22.2.8	uNomSpeed	45
5.23	entype_settings_t Struct Reference	45
5.23.1	Detailed Description	45
5.23.2	Field Documentation	45
5.23.2.1	DriverType	45
5.23.2.2	EngineType	46
5.24	extended_settings_t Struct Reference	46
5.24.1	Detailed Description	46
5.25	extio_settings_t Struct Reference	46
5.25.1	Detailed Description	47
5.25.2	Field Documentation	47
5.25.2.1	EXTIOModeFlags	47
5.25.2.2	EXTIOSetupFlags	47
5.26	feedback_settings_t Struct Reference	47
5.26.1	Detailed Description	48
5.26.2	Field Documentation	48
5.26.2.1	CountsPerTurn	48
5.26.2.2	FeedbackFlags	48
5.26.2.3	FeedbackType	48
5.26.2.4	IPS	48
5.27	gear_information_t Struct Reference	48
5.27.1	Detailed Description	49
5.27.2	Field Documentation	49
5.27.2.1	Manufacturer	49
5.27.2.2	PartNumber	49
5.28	gear_settings_t Struct Reference	49
5.28.1	Detailed Description	50
5.28.2	Field Documentation	50
5.28.2.1	Efficiency	50
5.28.2.2	InputInertia	50
5.28.2.3	MaxOutputBacklash	51
5.28.2.4	RatedInputSpeed	51
5.28.2.5	RatedInputTorque	51
5.28.2.6	ReductionIn	51
5.28.2.7	ReductionOut	51
5.29	get_position_calb_t Struct Reference	51
5.29.1	Detailed Description	52

5.29.2 Field Documentation	52
5.29.2.1 EncPosition	52
5.29.2.2 Position	52
5.30 get_position_t Struct Reference	52
5.30.1 Detailed Description	53
5.30.2 Field Documentation	53
5.30.2.1 EncPosition	53
5.30.2.2 uPosition	53
5.31 globally_unique_identifier_t Struct Reference	53
5.31.1 Detailed Description	54
5.31.2 Field Documentation	54
5.31.2.1 UniqueID0	54
5.31.2.2 UniqueID1	54
5.31.2.3 UniqueID2	54
5.31.2.4 UniqueID3	54
5.32 hallsensor_information_t Struct Reference	54
5.32.1 Detailed Description	55
5.32.2 Field Documentation	55
5.32.2.1 Manufacturer	55
5.32.2.2 PartNumber	55
5.33 hallsensor_settings_t Struct Reference	55
5.33.1 Detailed Description	56
5.33.2 Field Documentation	56
5.33.2.1 MaxCurrentConsumption	56
5.33.2.2 MaxOperatingFrequency	56
5.33.2.3 SupplyVoltageMax	57
5.33.2.4 SupplyVoltageMin	57
5.34 home_settings_calb_t Struct Reference	57
5.34.1 Detailed Description	57
5.34.2 Field Documentation	58
5.34.2.1 FastHome	58
5.34.2.2 HomeDelta	58
5.34.2.3 HomeFlags	58
5.34.2.4 SlowHome	58
5.35 home_settings_t Struct Reference	58
5.35.1 Detailed Description	59
5.35.2 Field Documentation	59
5.35.2.1 FastHome	59
5.35.2.2 HomeDelta	59
5.35.2.3 HomeFlags	59
5.35.2.4 SlowHome	59

5.35.2.5 uFastHome	60
5.35.2.6 uHomeDelta	60
5.35.2.7 uSlowHome	60
5.36 init_random_t Struct Reference	60
5.36.1 Detailed Description	60
5.36.2 Field Documentation	61
5.36.2.1 key	61
5.37 joystick_settings_t Struct Reference	61
5.37.1 Detailed Description	61
5.37.2 Field Documentation	62
5.37.2.1 DeadZone	62
5.37.2.2 ExpFactor	62
5.37.2.3 JoyCenter	62
5.37.2.4 JoyFlags	62
5.37.2.5 JoyHighEnd	62
5.37.2.6 JoyLowEnd	62
5.38 measurements_t Struct Reference	63
5.38.1 Detailed Description	63
5.38.2 Field Documentation	63
5.38.2.1 Error	63
5.38.2.2 Length	63
5.39 motor_information_t Struct Reference	64
5.39.1 Detailed Description	64
5.39.2 Field Documentation	64
5.39.2.1 Manufacturer	64
5.39.2.2 PartNumber	64
5.40 motor_settings_t Struct Reference	65
5.40.1 Detailed Description	66
5.40.2 Field Documentation	66
5.40.2.1 DetentTorque	66
5.40.2.2 MaxCurrent	66
5.40.2.3 MaxCurrentTime	66
5.40.2.4 MaxSpeed	67
5.40.2.5 MechanicalTimeConstant	67
5.40.2.6 MotorType	67
5.40.2.7 NoLoadCurrent	67
5.40.2.8 NoLoadSpeed	67
5.40.2.9 NominalCurrent	67
5.40.2.10 NominalPower	68
5.40.2.11 NominalSpeed	68
5.40.2.12 NominalTorque	68

5.40.2.13 NominalVoltage	68
5.40.2.14 Phases	68
5.40.2.15 Poles	68
5.40.2.16 RotorInertia	69
5.40.2.17 SpeedConstant	69
5.40.2.18 SpeedTorqueGradient	69
5.40.2.19 StallTorque	69
5.40.2.20 TorqueConstant	69
5.40.2.21 WindingInductance	69
5.40.2.22 WindingResistance	70
5.41 move_settings_calb_t Struct Reference	70
5.41.1 Detailed Description	70
5.41.2 Field Documentation	70
5.41.2.1 Accel	70
5.41.2.2 AntiplaySpeed	71
5.41.2.3 Decel	71
5.41.2.4 MoveFlags	71
5.41.2.5 Speed	71
5.42 move_settings_t Struct Reference	71
5.42.1 Detailed Description	72
5.42.2 Field Documentation	72
5.42.2.1 Accel	72
5.42.2.2 AntiplaySpeed	72
5.42.2.3 Decel	73
5.42.2.4 MoveFlags	73
5.42.2.5 Speed	73
5.42.2.6 uAntiplaySpeed	73
5.42.2.7 uSpeed	73
5.43 network_settings_t Struct Reference	73
5.43.1 Detailed Description	74
5.43.2 Field Documentation	74
5.43.2.1 DefaultGateway	74
5.43.2.2 DHCPEnabled	74
5.43.2.3 IPv4Address	74
5.43.2.4 SubnetMask	75
5.44 nonvolatile_memory_t Struct Reference	75
5.44.1 Detailed Description	75
5.44.2 Field Documentation	75
5.44.2.1 UserData	75
5.45 password_settings_t Struct Reference	75
5.45.1 Detailed Description	76

5.45.2 Field Documentation	76
5.45.2.1 UserPassword	76
5.46 pid_settings_t Struct Reference	76
5.46.1 Detailed Description	77
5.47 power_settings_t Struct Reference	77
5.47.1 Detailed Description	77
5.47.2 Field Documentation	78
5.47.2.1 CurrentSetTime	78
5.47.2.2 CurrReductDelay	78
5.47.2.3 HoldCurrent	78
5.47.2.4 PowerFlags	78
5.47.2.5 PowerOffDelay	78
5.48 secure_settings_t Struct Reference	78
5.48.1 Detailed Description	79
5.48.2 Field Documentation	79
5.48.2.1 Criticalpwr	79
5.48.2.2 Criticalusb	79
5.48.2.3 CriticalT	80
5.48.2.4 CriticalUpwr	80
5.48.2.5 CriticalUusb	80
5.48.2.6 Flags	80
5.48.2.7 LowUpwrOff	80
5.48.2.8 MinimumUusb	80
5.49 serial_number_t Struct Reference	80
5.49.1 Detailed Description	81
5.49.2 Field Documentation	81
5.49.2.1 Key	81
5.49.2.2 Major	81
5.49.2.3 Minor	81
5.49.2.4 Release	82
5.49.2.5 SN	82
5.50 set_position_calb_t Struct Reference	82
5.50.1 Detailed Description	82
5.50.2 Field Documentation	82
5.50.2.1 EncPosition	82
5.50.2.2 PosFlags	83
5.50.2.3 Position	83
5.51 set_position_t Struct Reference	83
5.51.1 Detailed Description	83
5.51.2 Field Documentation	83
5.51.2.1 EncPosition	83

5.51.2.2 PosFlags	84
5.51.2.3 uPosition	84
5.52 stage_information_t Struct Reference	84
5.52.1 Detailed Description	84
5.52.2 Field Documentation	84
5.52.2.1 Manufacturer	84
5.52.2.2 PartNumber	85
5.53 stage_name_t Struct Reference	85
5.53.1 Detailed Description	85
5.53.2 Field Documentation	85
5.53.2.1 PositionerName	85
5.54 stage_settings_t Struct Reference	85
5.54.1 Detailed Description	86
5.54.2 Field Documentation	86
5.54.2.1 HorizontalLoadCapacity	86
5.54.2.2 LeadScrewPitch	86
5.54.2.3 MaxCurrentConsumption	87
5.54.2.4 MaxSpeed	87
5.54.2.5 SupplyVoltageMax	87
5.54.2.6 SupplyVoltageMin	87
5.54.2.7 TravelRange	87
5.54.2.8 Units	87
5.54.2.9 VerticalLoadCapacity	88
5.55 status_calb_t Struct Reference	88
5.55.1 Detailed Description	89
5.55.2 Field Documentation	89
5.55.2.1 CmdBufFreeSpace	89
5.55.2.2 CurPosition	89
5.55.2.3 CurSpeed	89
5.55.2.4 CurT	89
5.55.2.5 EncPosition	89
5.55.2.6 EncSts	90
5.55.2.7 Flags	90
5.55.2.8 GPIOFlags	90
5.55.2.9 lpwr	90
5.55.2.10 lusb	90
5.55.2.11 MoveSts	90
5.55.2.12 MvCmdSts	90
5.55.2.13 PWRSts	91
5.55.2.14 Upwr	91
5.55.2.15 Uusb	91

5.55.2.16 WindSts	91
5.56 status_t Struct Reference	91
5.56.1 Detailed Description	92
5.56.2 Field Documentation	92
5.56.2.1 CmdBufFreeSpace	92
5.56.2.2 CurPosition	92
5.56.2.3 CurSpeed	93
5.56.2.4 CurT	93
5.56.2.5 EncPosition	93
5.56.2.6 EncSts	93
5.56.2.7 Flags	93
5.56.2.8 GPIOFlags	93
5.56.2.9 lpwr	93
5.56.2.10 lusb	94
5.56.2.11 MoveSts	94
5.56.2.12 MvCmdSts	94
5.56.2.13 PWRSts	94
5.56.2.14 uCurPosition	94
5.56.2.15 uCurSpeed	94
5.56.2.16 Upwr	94
5.56.2.17 Uusb	95
5.56.2.18 WindSts	95
5.57 sync_in_settings_calb_t Struct Reference	95
5.57.1 Detailed Description	95
5.57.2 Field Documentation	96
5.57.2.1 ClutterTime	96
5.57.2.2 Position	96
5.57.2.3 Speed	96
5.57.2.4 SyncInFlags	96
5.58 sync_in_settings_t Struct Reference	96
5.58.1 Detailed Description	97
5.58.2 Field Documentation	97
5.58.2.1 ClutterTime	97
5.58.2.2 Speed	97
5.58.2.3 SyncInFlags	97
5.58.2.4 uPosition	97
5.58.2.5 uSpeed	98
5.59 sync_out_settings_calb_t Struct Reference	98
5.59.1 Detailed Description	98
5.59.2 Field Documentation	98
5.59.2.1 Accuracy	98

5.59.2.2 SyncOutFlags	99
5.59.2.3 SyncOutPeriod	99
5.59.2.4 SyncOutPulseSteps	99
5.60 sync_out_settings_t Struct Reference	99
5.60.1 Detailed Description	100
5.60.2 Field Documentation	100
5.60.2.1 Accuracy	100
5.60.2.2 SyncOutFlags	100
5.60.2.3 SyncOutPeriod	100
5.60.2.4 SyncOutPulseSteps	100
5.60.2.5 uAccuracy	101
5.61 uart_settings_t Struct Reference	101
5.61.1 Detailed Description	101
5.61.2 Field Documentation	101
5.61.2.1 UARTSetupFlags	101
6 File Documentation	102
6.1 ximc.h File Reference	102
6.1.1 Detailed Description	127
6.1.2 Macro Definition Documentation	127
6.1.2.1 ALARM_ON_DRIVER_OVERHEATING	127
6.1.2.2 BACK_EMF_INDUCTANCE_AUTO	127
6.1.2.3 BACK_EMF_KM_AUTO	127
6.1.2.4 BACK_EMF_RESISTANCE_AUTO	127
6.1.2.5 BORDER_IS_ENCODER	127
6.1.2.6 BORDER_STOP_LEFT	127
6.1.2.7 BORDER_STOP_RIGHT	128
6.1.2.8 BORDERS_SWAP_MISSET_DETECTION	128
6.1.2.9 BRAKE_ENABLED	128
6.1.2.10 BRAKE_ENG_PWROFF	128
6.1.2.11 BRAKING_OVERVOLTAGE_PROTECTION	128
6.1.2.12 CONTROL_BTN_LEFT_PUSHED_OPEN	128
6.1.2.13 CONTROL_BTN_RIGHT_PUSHED_OPEN	128
6.1.2.14 CONTROL_MODE_BITS	129
6.1.2.15 CONTROL_MODE_JOY	129
6.1.2.16 CONTROL_MODE_LR	129
6.1.2.17 CONTROL_MODE_OFF	129
6.1.2.18 CTP_ALARM_ON_ERROR	129
6.1.2.19 CTP_BASE	129
6.1.2.20 CTP_ENABLED	129
6.1.2.21 CTP_ERROR_CORRECTION	130

6.1.2.22 DRIVER_TYPE_EXTERNAL	130
6.1.2.23 DRIVER_TYPE_INTEGRATE	130
6.1.2.24 EEPROM_PRECEDENCE	130
6.1.2.25 ENC_STATE_ABSENT	130
6.1.2.26 ENC_STATE_MALFUNC	130
6.1.2.27 ENC_STATE_OK	130
6.1.2.28 ENC_STATE_REVERS	131
6.1.2.29 ENC_STATE_UNKNOWN	131
6.1.2.30 ENDER_SW1_ACTIVE_LOW	131
6.1.2.31 ENDER_SW2_ACTIVE_LOW	131
6.1.2.32 ENDER_SWAP	131
6.1.2.33 ENGINE_ACCEL_ON	131
6.1.2.34 ENGINE_ANTIPLAY	131
6.1.2.35 ENGINE_CURRENT_AS_RMS	132
6.1.2.36 ENGINE_LIMIT_CURR	132
6.1.2.37 ENGINE_LIMIT_RPM	132
6.1.2.38 ENGINE_LIMIT_VOLT	132
6.1.2.39 ENGINE_MAX_SPEED	132
6.1.2.40 ENGINE_REVERSE	132
6.1.2.41 ENGINE_TYPE_2DC	133
6.1.2.42 ENGINE_TYPE_BRUSHLESS	133
6.1.2.43 ENGINE_TYPE_DC	133
6.1.2.44 ENGINE_TYPE_NONE	133
6.1.2.45 ENGINE_TYPE_STEP	133
6.1.2.46 ENGINE_TYPE_TEST	133
6.1.2.47 ENUMERATE_PROBE	133
6.1.2.48 EXTIO_SETUP_INVERT	134
6.1.2.49 EXTIO_SETUP_MODE_IN_ALARM	134
6.1.2.50 EXTIO_SETUP_MODE_IN_BITS	134
6.1.2.51 EXTIO_SETUP_MODE_IN_HOME	134
6.1.2.52 EXTIO_SETUP_MODE_IN_MOVR	134
6.1.2.53 EXTIO_SETUP_MODE_IN_NOP	134
6.1.2.54 EXTIO_SETUP_MODE_IN_PWOF	134
6.1.2.55 EXTIO_SETUP_MODE_IN_STOP	135
6.1.2.56 EXTIO_SETUP_MODE_OUT_ALARM	135
6.1.2.57 EXTIO_SETUP_MODE_OUT_BITS	135
6.1.2.58 EXTIO_SETUP_MODE_OUT_MOTOR_ON	135
6.1.2.59 EXTIO_SETUP_MODE_OUT_MOVING	135
6.1.2.60 EXTIO_SETUP_MODE_OUT_OFF	135
6.1.2.61 EXTIO_SETUP_MODE_OUT_ON	135
6.1.2.62 EXTIO_SETUP_OUTPUT	136

6.1.2.63 FEEDBACK_EMF	136
6.1.2.64 FEEDBACK_ENC_ADAPTIVE_HOLDING	136
6.1.2.65 FEEDBACK_ENC_FILTER_BITS	136
6.1.2.66 FEEDBACK_ENC_FILTER_MEDIUM	136
6.1.2.67 FEEDBACK_ENC_FILTER_NONE	136
6.1.2.68 FEEDBACK_ENC_FILTER_STRONG	136
6.1.2.69 FEEDBACK_ENC_FILTER_WEAK	137
6.1.2.70 FEEDBACK_ENC_REVERSE	137
6.1.2.71 FEEDBACK_ENC_TYPE_AUTO	137
6.1.2.72 FEEDBACK_ENC_TYPE_BITS	137
6.1.2.73 FEEDBACK_ENC_TYPE_DIFFERENTIAL	137
6.1.2.74 FEEDBACK_ENC_TYPE_SINGLE_ENDED	137
6.1.2.75 FEEDBACK_ENCODER	137
6.1.2.76 FEEDBACK_ENCODER_MEDIATED	138
6.1.2.77 FEEDBACK_NONE	138
6.1.2.78 H_BRIDGE_ALERT	138
6.1.2.79 HOME_DIR_FIRST	138
6.1.2.80 HOME_DIR_SECOND	138
6.1.2.81 HOME_HALF_MV	138
6.1.2.82 HOME_MV_SEC_EN	138
6.1.2.83 HOME_STOP_FIRST_BITS	139
6.1.2.84 HOME_STOP_FIRST_LIM	139
6.1.2.85 HOME_STOP_FIRST_REV	139
6.1.2.86 HOME_STOP_FIRST_SYN	139
6.1.2.87 HOME_STOP_SECOND_BITS	139
6.1.2.88 HOME_STOP_SECOND_LIM	139
6.1.2.89 HOME_STOP_SECOND_REV	139
6.1.2.90 HOME_STOP_SECOND_SYN	140
6.1.2.91 HOME_USE_FAST	140
6.1.2.92 JOY_REVERSE	140
6.1.2.93 LOW_UPWR_PROTECTION	140
6.1.2.94 MICROSTEP_MODE_FRAC_128	140
6.1.2.95 MICROSTEP_MODE_FRAC_16	140
6.1.2.96 MICROSTEP_MODE_FRAC_2	140
6.1.2.97 MICROSTEP_MODE_FRAC_256	141
6.1.2.98 MICROSTEP_MODE_FRAC_32	141
6.1.2.99 MICROSTEP_MODE_FRAC_4	141
6.1.2.100 MICROSTEP_MODE_FRAC_64	141
6.1.2.101 MICROSTEP_MODE_FRAC_8	141
6.1.2.102 MICROSTEP_MODE_FULL	141
6.1.2.103 MOVE_STATE_ANTIPLAY	141

6.1.2.104 MOVE_STATE_MOVING	142
6.1.2.105 MOVE_STATE_TARGET_SPEED	142
6.1.2.106 MVCMD_ERROR	142
6.1.2.107 MVCMD_HOME	142
6.1.2.108 MVCMD_LEFT	142
6.1.2.109 MVCMD_LOFT	142
6.1.2.110 MVCMD_MOVE	142
6.1.2.111 MVCMD_MOVR	143
6.1.2.112 MVCMD_NAME_BITS	143
6.1.2.113 MVCMD_RIGHT	143
6.1.2.114 MVCMD_RUNNING	143
6.1.2.115 MVCMD_SSTP	143
6.1.2.116 MVCMD_STOP	143
6.1.2.117 MVCMD_UKNWN	143
6.1.2.118 POWER_OFF_ENABLED	144
6.1.2.119 POWER_REDUCT_ENABLED	144
6.1.2.120 POWER_SMOOTH_CURRENT	144
6.1.2.121 PWR_STATE_MAX	144
6.1.2.122 PWR_STATE_NORM	144
6.1.2.123 PWR_STATE_OFF	144
6.1.2.124 PWR_STATE_REDUCT	144
6.1.2.125 PWR_STATE_UNKNOWN	145
6.1.2.126 REV_SENS_INV	145
6.1.2.127 RPM_DIV_1000	145
6.1.2.128 SETPOS_IGNORE_ENCODER	145
6.1.2.129 SETPOS_IGNORE_POSITION	145
6.1.2.130 STATE_ALARM	145
6.1.2.131 STATE_BORDERS_SWAP_MISSET	145
6.1.2.132 STATE_BRAKE	146
6.1.2.133 STATE_BUTTON_LEFT	146
6.1.2.134 STATE_BUTTON_RIGHT	146
6.1.2.135 STATE_CONTR	146
6.1.2.136 STATE_CONTROLLER_OVERHEAT	146
6.1.2.137 STATE_CTP_ERROR	146
6.1.2.138 STATE_DIG_SIGNAL	146
6.1.2.139 STATE_EEPROM_CONNECTED	147
6.1.2.140 STATE_ENC_A	147
6.1.2.141 STATE_ENC_B	147
6.1.2.142 STATE_ENGINE_RESPONSE_ERROR	147
6.1.2.143 STATE_ERRC	147
6.1.2.144 STATE_ERRD	147

6.1.2.145 STATE_ERRV	148
6.1.2.146 STATE_EXTIO_ALARM	148
6.1.2.147 STATE_GPIO_LEVEL	148
6.1.2.148 STATE_GPIO_PINOUT	148
6.1.2.149 STATE_IS_HOMED	148
6.1.2.150 STATE_LEFT_EDGE	148
6.1.2.151 STATE_LOW_USB_VOLTAGE	148
6.1.2.152 STATE_OVERLOAD_POWER_CURRENT	149
6.1.2.153 STATE_OVERLOAD_POWER_VOLTAGE	149
6.1.2.154 STATE_OVERLOAD_USB_CURRENT	149
6.1.2.155 STATE_OVERLOAD_USB_VOLTAGE	149
6.1.2.156 STATE_POWER_OVERHEAT	149
6.1.2.157 STATE_REV_SENSOR	149
6.1.2.158 STATE_RIGHT_EDGE	150
6.1.2.159 STATE_SECUR	150
6.1.2.160 STATE_SYNC_INPUT	150
6.1.2.161 STATE_SYNC_OUTPUT	150
6.1.2.162 STATE_WINDING_RES_MISMATCH	150
6.1.2.163 SYNCIN_ENABLED	150
6.1.2.164 SYNCIN_GOTOPOSITION	150
6.1.2.165 SYNCIN_INVERT	151
6.1.2.166 SYNCOUT_ENABLED	151
6.1.2.167 SYNCOUT_IN_STEPS	151
6.1.2.168 SYNCOUT_INVERT	151
6.1.2.169 SYNCOUT_ONPERIOD	151
6.1.2.170 SYNCOUT_ONSTART	151
6.1.2.171 SYNCOUT_ONSTOP	151
6.1.2.172 SYNCOUT_STATE	152
6.1.2.173 UART_PARITY_BITS	152
6.1.2.174 WIND_A_STATE_ABSENT	152
6.1.2.175 WIND_A_STATE_MALFUNC	152
6.1.2.176 WIND_A_STATE_OK	152
6.1.2.177 WIND_A_STATE_UNKNOWN	152
6.1.2.178 WIND_B_STATE_ABSENT	152
6.1.2.179 WIND_B_STATE_MALFUNC	153
6.1.2.180 WIND_B_STATE_OK	153
6.1.2.181 WIND_B_STATE_UNKNOWN	153
6.1.2.182 XIMC_API	153
6.1.2.183 XIMC_CALLCONV	153
6.1.2.184 XIMC_RETTYPE	153
6.1.3 Typedef Documentation	154

6.1.3.1 calibration_t	154
6.1.3.2 device_network_information_t	154
6.1.3.3 logging_callback_t	154
6.1.4 Function Documentation	155
6.1.4.1 close_device()	155
6.1.4.2 command_clear_fram()	155
6.1.4.3 command_eeread_settings()	155
6.1.4.4 command_eesave_settings()	156
6.1.4.5 command_home()	156
6.1.4.6 command_homezero()	156
6.1.4.7 command_left()	157
6.1.4.8 command_loft()	157
6.1.4.9 command_move()	157
6.1.4.10 command_move_calb()	158
6.1.4.11 command_movr()	158
6.1.4.12 command_movr_calb()	158
6.1.4.13 command_power_off()	159
6.1.4.14 command_read_robust_settings()	159
6.1.4.15 command_read_settings()	159
6.1.4.16 command_reset()	160
6.1.4.17 command_right()	160
6.1.4.18 command_save_robust_settings()	160
6.1.4.19 command_save_settings()	160
6.1.4.20 command_sstp()	161
6.1.4.21 command_start_measurements()	161
6.1.4.22 command_stop()	161
6.1.4.23 command_update_firmware()	161
6.1.4.24 command_wait_for_stop()	162
6.1.4.25 command_zero()	162
6.1.4.26 enumerate_devices()	163
6.1.4.27 free_enumerate_devices()	164
6.1.4.28 get_accessories_settings()	165
6.1.4.29 get_analog_data()	165
6.1.4.30 get_bootloader_version()	165
6.1.4.31 get_brake_settings()	166
6.1.4.32 get_calibration_settings()	166
6.1.4.33 get_chart_data()	166
6.1.4.34 get_control_settings()	167
6.1.4.35 get_control_settings_calb()	167
6.1.4.36 get_controller_name()	168
6.1.4.37 get_ctp_settings()	168

6.1.4.38	get_debug_read()	168
6.1.4.39	get_device_count()	169
6.1.4.40	get_device_information()	169
6.1.4.41	get_device_name()	169
6.1.4.42	get_edges_settings()	170
6.1.4.43	get_edges_settings_calb()	170
6.1.4.44	get_emf_settings()	171
6.1.4.45	get_encoder_information()	171
6.1.4.46	get_encoder_settings()	171
6.1.4.47	get_engine_advanced_setup()	172
6.1.4.48	get_engine_settings()	172
6.1.4.49	get_engine_settings_calb()	172
6.1.4.50	get_entype_settings()	173
6.1.4.51	get_enumerate_device_controller_name()	173
6.1.4.52	get_enumerate_device_information()	173
6.1.4.53	get_enumerate_device_network_information()	174
6.1.4.54	get_enumerate_device_serial()	174
6.1.4.55	get_enumerate_device_stage_name()	174
6.1.4.56	get_extended_settings()	175
6.1.4.57	get_extio_settings()	175
6.1.4.58	get_feedback_settings()	175
6.1.4.59	get_firmware_version()	176
6.1.4.60	get_gear_information()	176
6.1.4.61	get_gear_settings()	176
6.1.4.62	get_globally_unique_identifier()	177
6.1.4.63	get_hallsensor_information()	177
6.1.4.64	get_hallsensor_settings()	177
6.1.4.65	get_home_settings()	178
6.1.4.66	get_home_settings_calb()	178
6.1.4.67	get_init_random()	178
6.1.4.68	get_joystick_settings()	179
6.1.4.69	get_measurements()	179
6.1.4.70	get_motor_information()	180
6.1.4.71	get_motor_settings()	180
6.1.4.72	get_move_settings()	180
6.1.4.73	get_move_settings_calb()	180
6.1.4.74	get_network_settings()	181
6.1.4.75	get_nonvolatile_memory()	181
6.1.4.76	get_password_settings()	181
6.1.4.77	get_pid_settings()	182
6.1.4.78	get_position()	182

6.1.4.79 get_position_calb()	182
6.1.4.80 get_power_settings()	183
6.1.4.81 get_secure_settings()	183
6.1.4.82 get_serial_number()	183
6.1.4.83 get_stage_information()	184
6.1.4.84 get_stage_name()	184
6.1.4.85 get_stage_settings()	184
6.1.4.86 get_status()	184
6.1.4.87 get_status_calb()	185
6.1.4.88 get_sync_in_settings()	185
6.1.4.89 get_sync_in_settings_calb()	186
6.1.4.90 get_sync_out_settings()	186
6.1.4.91 get_sync_out_settings_calb()	186
6.1.4.92 get_uart_settings()	187
6.1.4.93 goto_firmware()	187
6.1.4.94 has_firmware()	187
6.1.4.95 logging_callback_stderr_narrow()	188
6.1.4.96 logging_callback_stderr_wide()	188
6.1.4.97 msec_sleep()	188
6.1.4.98 open_device()	188
6.1.4.99 probe_device()	189
6.1.4.100 reset_locks()	189
6.1.4.101 service_command_updf()	189
6.1.4.102 set_accessories_settings()	189
6.1.4.103 set_brake_settings()	190
6.1.4.104 set_calibration_settings()	190
6.1.4.105 set_control_settings()	191
6.1.4.106 set_control_settings_calb()	192
6.1.4.107 set_controller_name()	192
6.1.4.108 set_correction_table()	192
6.1.4.109 set_ctp_settings()	193
6.1.4.110 set_debug_write()	193
6.1.4.111 set_edges_settings()	194
6.1.4.112 set_edges_settings_calb()	194
6.1.4.113 set_emf_settings()	195
6.1.4.114 set_encoder_information()	195
6.1.4.115 set_encoder_settings()	195
6.1.4.116 set_engine_advanced_setup()	196
6.1.4.117 set_engine_settings()	196
6.1.4.118 set_engine_settings_calb()	196
6.1.4.119 set_entype_settings()	197

6.1.4.120	set_extended_settings()	197
6.1.4.121	set_extio_settings()	197
6.1.4.122	set_feedback_settings()	198
6.1.4.123	set_gear_information()	198
6.1.4.124	set_gear_settings()	198
6.1.4.125	set_hallsensor_information()	199
6.1.4.126	set_hallsensor_settings()	199
6.1.4.127	set_home_settings()	199
6.1.4.128	set_home_settings_calb()	200
6.1.4.129	set_joystick_settings()	200
6.1.4.130	set_logging_callback()	201
6.1.4.131	set_motor_information()	201
6.1.4.132	set_motor_settings()	201
6.1.4.133	set_move_settings()	201
6.1.4.134	set_move_settings_calb()	202
6.1.4.135	set_network_settings()	202
6.1.4.136	set_nonvolatile_memory()	202
6.1.4.137	set_password_settings()	203
6.1.4.138	set_pid_settings()	203
6.1.4.139	set_position()	203
6.1.4.140	set_position_calb()	204
6.1.4.141	set_power_settings()	204
6.1.4.142	set_secure_settings()	204
6.1.4.143	set_serial_number()	205
6.1.4.144	set_stage_information()	205
6.1.4.145	set_stage_name()	205
6.1.4.146	set_stage_settings()	206
6.1.4.147	set_sync_in_settings()	206
6.1.4.148	set_sync_in_settings_calb()	206
6.1.4.149	set_sync_out_settings()	207
6.1.4.150	set_sync_out_settings_calb()	207
6.1.4.151	set_uart_settings()	208
6.1.4.152	write_key()	208
6.1.4.153	ximc_version()	208
6.2	ximc.h	209

Chapter 1

libximc library

Documentation for libximc library.

Libximc is **thread safe**, cross-platform library for working with 8SMC4-USB and 8SMC5-USB controllers.

Full documentation about controllers is [there](#).

Full documentation about libximc API is available on the page [ximc.h](#).

The libximc library is now available on [PyPI](#), and can be installed directly using pip:

```
pip install libximc
```

This simplifies the use of the library in Python projects without the need for manual building.

1.1 What the controller does

- Supports input and output synchronization signals to ensure the joint operation of multiple devices within a complex system ;.
- Works with all compact stepper motors with a winding current of up to 3 A, without feedback, as well as with stepper motors equipped with an encoder in the feedback circuit, including a linear encoder on the positioner.
- Manages controller using ready-made [xilab software](#) or using examples which allow rapid development using C++, C#, .NET, Visual Basic, Xcode, Python, Matlab, Java and and LabVIEW.

1.2 What can do libximc library

- Libximc manages controller using interfaces: USB 2.0, RS232 and Ethernet, also uses a common and proven virtual serial port interface, so you can work with motor control modules through this library under almost all operating systems, including Windows, Linux and macOS
- Libximc library supports plug/unplug on the fly. Each device can be controlled only by one program at once. **Multiple processes (programs) that control one device simultaneously are not allowed!**

Warning

Libximc library opens the controller in exclusive access mode. Any controller opened with libximc (Xi↔ Lab also uses this library) needs to be closed before it may be used by another process. So at first check that you have closed XILab or other software dealing with the controller before trying to reopen the controller.

Please read the [Introduction](#) to start work with library.

To use libximc in your project please consult with [How to use with...](#)

1.3 Assistance

Many thanks to everyone who sends us **errors** and **suggestions**. We appreciate your suggestions and try to make our product better!

Chapter 2

Introduction

2.1 About library

This document contains all the information about the libximc library, except for the building instructions, which you can find in the README.md file at the root of the GitHub repository: <https://github.com/↔Standa-Optomechanics/libximc>. It utilizes well known virtual COM-port interface, so you can use it on Windows, Linux, macOS for Intel and Apple Silicon (via Rosetta 2) including 64-bit versions. Multi-platform programming library supports plug/unplug on the fly.

Each device can be controlled only by one program at once. Multiple processes (programs) that control one device simultaneously are not allowed.

2.1.1 Supported OS and environment requirements:

- macOS 10.15 or newer
- Windows 7 or newer
- Linux debian-based

Chapter 3

How to use with...

To acquire the first skills of using the library, a simple `testappeasy_C` test application has been created. Languages other than C are supported using calls with conversion of arguments of the `stdcall` type. A simple C test application is located in the `'examples/test_C'` directory, a C# project is located in `'examples/test_CSharp'`, on VB.NET - in `'examples/test_VBNET'`, for matlab - `'examples/test_MATLAB'`, for Java - `'examples/test_Java'`, for Python - `'examples/test_Python'`. Libraries, header files and other necessary files are located in the directories `'ximc/win32'`, `'ximc/win64'`, `'ximc/macosx'` and the like. The developer kit also includes already compiled examples: `testapp` and `testappeasy` x32 and x64 bits for windows and only x64 bits for macOS, `test_CSharp`, `test_VBNET`, `test_Java` - cross-platform, `test_MATLAB` and `test_Python` do not require compilation.

Note

SDK requires Microsoft Visual C++ Redistributable Package (provided with SDK - `vcredist_x86` or `vcredist_x64`)

On Linux both the `libximc7_x.x.x` and `libximc7-dev_x.x.x` target architecture in the specified order. For install packages, you can use the `.deb` command: `dpkg -i filename.deb`, where `filename.deb` is the name of the package (packages in Debian have the extension `.deb`). You must run `dpkg` with superuser privileges (root).

3.0.1 Visual C++

`Testapp` can be built using `testapp.sln`. Library must be compiled with MS Visual C++ too, `mingw-library`. Make sure that Microsoft Visual C++ Redistributable Package is installed.

Open solution `examples/testapp/testapp.sln`, build and run from the IDE.

In case of the 8SMC4-USB-Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in `testapp.c` file before build (see `enumerate_hints` variable).

3.0.2 CodeBlocks

`Testappeasy_C` and `testapp_C` can be built using `testappeasy_C.cbp` and `testapp_C.cbp` respectively. Library must be compiled with MS Visual C++ too, `mingw-library`. Make sure that Microsoft Visual C++ Redistributable Package is installed. *

Open solution `examples/test_C/testappeasy_C/testappeasy_C/testappeasy_C.cbp` or `examples/test_C/testapp_C/testapp_C/testapp_C.cbp`, build and run from the IDE.

3.0.3 MinGW

MinGW is a port of GCC to win32 platform. It's required to install MinGW package.

MinGW-compiled testapp can be built with MS Visual C++ or mingw library.

```
mingw32-make -f Makefile.mingw all
```

Then copy library libximc.dll to current directory and launch testapp.exe.

In case of the 8Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in testapp.c file before build (see enumerate_hints variable).

3.0.4 C++ Builder

First of all, you should create a library suitable for C++ Builder. **Visual C++ and Builder libraries are not compatible** Invoke:

```
implib libximc.lib libximc.def
```

Then compile test application:

```
bcc32 -I..\..\ximc\win32 -L..\..\ximc\win32 -DWIN32 -DNDEBUG -D_WINDOWS testapp.c libximc.lib
```

In case of the 8Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in testapp.c file before build (see enumerate_hints variable).

There is also an **unsupported example** of using libximc in a C++ Builder project

3.0.5 XCode

testapp should be built with XCode project testapp.xcodeproj. Library is a macOS framework, and at example application it's bundled inside testapp.app

Then launch application testapp.app and check activity output in Console.app.

In case of the 8Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in testapp.c file before build (see enumerate_hints variable). There is also **an example of using the libximc library** in a C++ Builder project, **but it is not supported**.

3.0.6 GCC

Make sure that libximc (rpm or deb) is installed at your system. Installation of package should be performed with a package manager of operating system. On macOS a framework is provided.

Note that user should belong to system group which allows access to a serial port (dip or serial, for example).

Test application can be built with the installed library with the following script:

```
make
```

In case of cross-compilation (target architecture differs from the current system architecture) feed -m64 or -m32 flag to compiler. On macOS it's needed to use -arch flag instead to build an universal binary. Please consult a compiler documentation.

Then launch the application as:

```
make run
```

Note: make run on macOS copies a library to the current directory. If you want to use library from the custom directory please be sure to specify LD_LIBRARY_PATH or DYLD_LIBRARY_PATH to the directory with the library.

In case of the 8SMC4-USB-Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in testapp.c file before build (see enumerate_hints variable).

3.0.7 .NET

Wrapper assembly for libximc.dll is ximc/winX/wrappers/csharp/ximcnet.dll. It is provided with two different architectures. Tested on platforms .NET from 2.0 to 4.5.1

Test .NET applications for Visual Studio 2013 is located at test_CSharp (for C#) and test_VBNET (for VB.NET) respectively. Open solutions and build it.

In case of the 8SMC4-USB-Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in testapp.cs or testapp.vb file (depending on programming language) before build (see enumerate_hints variable for C# or enum_hints variable for VB).

3.0.8 Java

How to run example on Linux. Go to to examples/test_Java/compiled-winX/ and run:

```
java -cp /usr/share/java/libximc.jar:test_Java.jar ru.ximc.TestJava
```

How to run example on Windows. Go to to examples/test_Java/compiled-winX/. Then run:

```
java -classpath libximc.jar -classpath test_Java.jar ru.ximc.TestJava
```

How to modify and recompile an example. Go to to examples/test_Java/compiled. Sources are embedded in a test_Java.jar. Extract them:

```
jar xvf test_Java.jar ru META-INF
```

Then rebuild sources:

```
javac -classpath /usr/share/java/libximc.jar -Xlint ru/ximc/TestJava.java
```

or for Windows or macOS

```
javac -classpath libximc.jar -Xlint ru/ximc/TestJava.java
```

Then build a jar:

```
jar cmf META-INF/MANIFEST.MF test_Java.jar ru
```

In case of the 8Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in TestJava.java file before build (see ENUM_HINTS variable).

3.0.9 Python

Change current directory to the examples/test_Python/xxxtest. NB: For libximc usage, the example uses the wrapper module ximc/crossplatform/wrappers/python/libximc.

To run:

```
python xxxx.py
```

In case of the 8Eth1 Ethernet adapter usage, it's necessary to set correct IP address of the Ethernet adapter in test_Python.py file before launch (see enum_hints variable).

3.0.10 MATLAB

Sample MATLAB program testximc.m is provided at the directory examples/test_MATLAB. On windows copy `ximc.h`, `libximc.dll`, `bindy.dll`, `xiwrapper.dll` and contents of `ximc/(win32,win64)/wrappers/matlab/` directory to the current directory.

Before launch:

On macOS: copy `ximc/macosx/libximc.framework`, `ximc/macosx/wrappers/ximcm.h`, `ximc/ximc.h` to the directory examples/test_MATLAB. Install XCode compatible with Matlab.

On Linux: install `libximc*deb` and `libximc-dev*dev` of target architecture. Then copy `ximc/macosx/wrappers/ximcm.h` to the directory examples/matlab. Install gcc compatible with Matlab.

For XCode and gcc version compatibility check document <https://www.mathworks.com/content/dam/mathworks/mathworks/SystemRequirements-Release2014a-SupportedCompilers.pdf> or similar.

On Windows before the start nothing needs to be done

Change current directory in the MATLAB to the examples/test_MATLAB. Then launch in MATLAB prompt:

```
testximc
```

In case of the 8Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in testximc.m file before launch (see enum_hints variable).

3.1 Generic logging facility

If you want to turn on file logging, you should run the program that uses libximc library with the "XILog" environment variable set to desired file name. This file will be opened for writing on the first log event and will be closed when the program which uses libximc terminates. Data which is sent to/received from the controller is logged along with port open and close events.

3.2 Required permissions

libximc generally does not require special permissions to work, it only needs read/write access to USB-serial ports on the system. An exception to this rule is a Windows-only "fix_usbser_sys()" function - it needs elevation and will produce null result if run as a regular user.

3.3 C-profiles

C-profiles are header files distributed with the libximc library. They enable one to set all controller settings for any of the supported stages with a single function call in a C/C++ program.

You may see how to use C-profiles in the example directory "examples/test_C/testprofile_C".

3.4 Python-profiles

Python-profiles are sets of configuration functions distributed with the libximc library. They allow to load the controller with settings of one of the supported stages using a single function call in a Python program.

You may see how to use Python-profiles in the example "examples/test_Python/profiletest/testpythonprofile.py".

Chapter 4

Working with user units

In addition to working in basic units(steps, encoder value), the library allows you to work with user units. For this purpose are used:

- The structure of the conversion units [calibration_t](#)
- The functions of which have doubles for working with user units, data structures for these functions
- Coordinate correction table for more accurate positioning

4.1 The structure of the conversion units `calibration_t`

To specify conversion of the basic units in the user and back, [calibration_t](#) structure is used. With the help of coefficients A and MicrostepMode, specified in this structure, steps and microsteps which are integers are converted into the user value of the real type and back.

Conversion formulas:

- The conversion to user units.

```
user_value = A*(step + mstep/pow(2, MicrostepMode-1))
```

- Conversion from user units.

```
step = (int)(user_value/A)
mstep = (user_value/A - step)*pow(2, MicrostepMode-1)
```

4.2 Alternative functions for working with user units and data structures for them

Structures and functions for working with user units have the `_calb` postfix. The user using these functions can perform all actions in their own units without worrying about the computations of the controller. The data format of `_calb` structures is described in detail. For `_calb` functions particular descriptions are not used. They perform the same actions as the basic functions do. The difference between them and the basic functions is in the position, velocity, and acceleration of the data types defined as user-defined. If clarification for `_calb` functions is necessary, they are provided as notes in the description of the basic functions.

4.3 Coordinate correction table for more accurate positioning

Some functions for working with user units support coordinate transformation using a correction table. To load a table from a file, the [set_correction_table\(\)](#) function is used. Its description contains the functions and their data supporting correction.

Note

For data fields which are corrected in case of loading of the table in the description of the field is written - corrected by the table.

File format:

- two columns separated by tabs;
- column headers are string;
- real type data, point is a separator;
- the first column is the coordinate, the second is the deviation caused by a mechanical error;
- the deviation between coordinates is calculated linearly;
- constant is equal to the deviation at the boundary beyond the range;
- maximum length of the table is 100 lines.

Sample file:

```
X dX
0 0
5.0 0.005
10.0 -0.01
```

Chapter 5

Data Structure Documentation

5.1 accessories_settings_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- char [MagneticBrakeInfo](#) [25]
The manufacturer and the part number of magnetic brake, the maximum string length is 24 characters.
- float [MBRatedVoltage](#)
Rated voltage for controlling the magnetic brake (V).
- float [MBRatedCurrent](#)
Rated current for controlling the magnetic brake (A).
- float [MBTorque](#)
*Retention moment (mN * m).*
- unsigned int [MBSettings](#)
Magnetic brake settings flags.
- char [TemperatureSensorInfo](#) [25]
The manufacturer and the part number of the temperature sensor, the maximum string length: 24 characters.
- float [TSMIn](#)
The minimum measured temperature (degrees Celsius) Data type: float.
- float [TSMaX](#)
The maximum measured temperature (degrees Celsius) Data type: float.
- float [TSGraD](#)
The temperature gradient (V/degrees Celsius).
- unsigned int [TSSettings](#)
Temperature sensor settings flags.
- unsigned int [LimitSwitchesSettings](#)
Temperature sensor settings flags.

5.1.1 Detailed Description

Deprecated.

Additional accessories' information.

See also

[set_accessories_settings](#)

[get_accessories_settings](#)

[get_accessories_settings](#), [set_accessories_settings](#)

5.1.2 Field Documentation

5.1.2.1 LimitSwitchesSettings

```
unsigned int LimitSwitchesSettings
```

[Temperature sensor settings flags](#).

5.1.2.2 MagneticBrakeInfo

```
char MagneticBrakeInfo[25]
```

The manufacturer and the part number of magnetic brake, the maximum string length is 24 characters.

5.1.2.3 MBRatedCurrent

```
float MBRatedCurrent
```

Rated current for controlling the magnetic brake (A).

Data type: float.

5.1.2.4 MBRatedVoltage

```
float MBRatedVoltage
```

Rated voltage for controlling the magnetic brake (V).

Data type: float.

5.1.2.5 MBSettings

```
unsigned int MBSettings
```

[Magnetic brake settings flags](#).

5.1.2.6 MBTorque

`float MBTorque`

Retention moment (mN * m).

Data type: float.

5.1.2.7 TemperatureSensorInfo

`char TemperatureSensorInfo[25]`

The manufacturer and the part number of the temperature sensor, the maximum string length: 24 characters.

5.1.2.8 TSGrad

`float TSGrad`

The temperature gradient (V/degrees Celsius).

Data type: float.

5.1.2.9 TSMax

`float TSMax`

The maximum measured temperature (degrees Celsius) Data type: float.

5.1.2.10 TSMin

`float TSMin`

The minimum measured temperature (degrees Celsius) Data type: float.

5.1.2.11 TSSettings

`unsigned int TSSettings`

[Temperature sensor settings flags.](#)

5.2 analog_data_t Struct Reference

Analog data.

```
#include <ximc.h>
```

Data Fields

- unsigned int [A1Voltage_ADC](#)
"Voltage on pin 1 winding A" raw data from ADC.
- unsigned int [A2Voltage_ADC](#)
"Voltage on pin 2 winding A" raw data from ADC.
- unsigned int [B1Voltage_ADC](#)
"Voltage on pin 1 winding B" raw data from ADC.
- unsigned int [B2Voltage_ADC](#)
"Voltage on pin 2 winding B" raw data from ADC.
- unsigned int [SupVoltage_ADC](#)
"Supply voltage of H-bridge's MOSFETs" raw data from ADC.
- unsigned int [ACurrent_ADC](#)
"Winding A current" raw data from ADC.
- unsigned int [BCurrent_ADC](#)
"Winding B current" raw data from ADC.
- unsigned int [FullCurrent_ADC](#)
"Full current" raw data from ADC.
- unsigned int [Temp_ADC](#)
Voltage from temperature sensor, raw data from ADC.
- unsigned int [Joy_ADC](#)
Joystick raw data from ADC.
- unsigned int **Pot_ADC**
Voltage on analog input, raw data from ADC
- unsigned int [Enc_Check_ADC](#)
Voltage on encoder check line, raw ADC data.
- unsigned int **deprecated0**
- int [A1Voltage](#)
"Voltage on pin 1 winding A" calibrated data (in tens of mV).
- int [A2Voltage](#)
"Voltage on pin 2 winding A" calibrated data (in tens of mV).
- int [B1Voltage](#)
"Voltage on pin 1 winding B" calibrated data (in tens of mV).
- int [B2Voltage](#)
"Voltage on pin 2 winding B" calibrated data (in tens of mV).
- int [SupVoltage](#)
"Supply voltage on the top of H-bridge's MOSFETs" calibrated data (in tens of mV).
- int [ACurrent](#)
"Winding A current" calibrated data (in mA).
- int [BCurrent](#)
"Winding B current" calibrated data (in mA).
- int [FullCurrent](#)
"Full current" calibrated data (in mA).
- int [Temp](#)
Temperature, calibrated data (in tenths of degrees Celsius).
- int [Joy](#)
Joystick, calibrated data.
- int [Pot](#)
Analog input, calibrated data.
- int [Enc_Check](#)

Voltage on encoder check line, exponentially filtrated ADC codes.

- unsigned int **deprecated1** [2]
- int [R](#)

Motor winding resistance in mOhms (is only used with stepper motors).

- int [L](#)

Motor winding pseudo inductance in uH (is only used with stepper motors).

5.2.1 Detailed Description

Analog data.

This structure contains raw analog data from the embedded ADC. These data are used for device testing and deep recalibration by the manufacturer only.

See also

[get_analog_data](#)

[get_analog_data](#)

5.2.2 Field Documentation

5.2.2.1 A1Voltage

int A1Voltage

"Voltage on pin 1 winding A" calibrated data (in tens of mV).

5.2.2.2 A1Voltage_ADC

unsigned int A1Voltage_ADC

"Voltage on pin 1 winding A" raw data from ADC.

5.2.2.3 A2Voltage

int A2Voltage

"Voltage on pin 2 winding A" calibrated data (in tens of mV).

5.2.2.4 A2Voltage_ADC

unsigned int A2Voltage_ADC

"Voltage on pin 2 winding A" raw data from ADC.

5.2.2.5 ACurrent

`int ACurrent`

"Winding A current" calibrated data (in mA).

5.2.2.6 ACurrent_ADC

`unsigned int ACurrent_ADC`

"Winding A current" raw data from ADC.

5.2.2.7 B1Voltage

`int B1Voltage`

"Voltage on pin 1 winding B" calibrated data (in tens of mV).

5.2.2.8 B1Voltage_ADC

`unsigned int B1Voltage_ADC`

"Voltage on pin 1 winding B" raw data from ADC.

5.2.2.9 B2Voltage

`int B2Voltage`

"Voltage on pin 2 winding B" calibrated data (in tens of mV).

5.2.2.10 B2Voltage_ADC

`unsigned int B2Voltage_ADC`

"Voltage on pin 2 winding B" raw data from ADC.

5.2.2.11 BCurrent

`int BCurrent`

"Winding B current" calibrated data (in mA).

5.2.2.12 BCurrent_ADC

`unsigned int BCurrent_ADC`

"Winding B current" raw data from ADC.

5.2.2.13 Enc_Check

`int Enc_Check`

Voltage on encoder check line, exponentially filtrated ADC codes.

Used to determine encoder type: single-ended or differential.

5.2.2.14 Enc_Check_ADC

`unsigned int Enc_Check_ADC`

Voltage on encoder check line, raw ADC data.

Used to determine encoder type: single-ended or differential.

5.2.2.15 FullCurrent

`int FullCurrent`

"Full current" calibrated data (in mA).

5.2.2.16 FullCurrent_ADC

`unsigned int FullCurrent_ADC`

"Full current" raw data from ADC.

5.2.2.17 Joy

`int Joy`

Joystick, calibrated data.

Range: 0..10000

5.2.2.18 Joy_ADC

`unsigned int Joy_ADC`

Joystick raw data from ADC.

5.2.2.19 L

`int L`

Motor winding pseudo inductance in uH (is only used with stepper motors).

5.2.2.20 Pot

`int Pot`

Analog input, calibrated data.

Range: 0..10000

5.2.2.21 R

`int R`

Motor winding resistance in mOhms (is only used with stepper motors).

5.2.2.22 SupVoltage

`int SupVoltage`

"Supply voltage on the top of H-bridge's MOSFETs" calibrated data (in tens of mV).

5.2.2.23 SupVoltage_ADC

`unsigned int SupVoltage_ADC`

"Supply voltage of H-bridge's MOSFETs" raw data from ADC.

5.2.2.24 Temp

`int Temp`

Temperature, calibrated data (in tenths of degrees Celsius).

5.2.2.25 Temp_ADC

`unsigned int Temp_ADC`

Voltage from temperature sensor, raw data from ADC.

5.3 brake_settings_t Struct Reference

Brake settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [t1](#)
Time in ms between turning on motor power and turning off the brake.
- unsigned int [t2](#)
Time in ms between the brake turning off and moving readiness.
- unsigned int [t3](#)
Time in ms between motor stop and the brake turning on.
- unsigned int [t4](#)
Time in ms between turning on the brake and turning off motor power.
- unsigned int [BrakeFlags](#)
Brake settings flags.

5.3.1 Detailed Description

Brake settings.

This structure contains brake control parameters.

See also

[set_brake_settings](#)
[get_brake_settings](#)
[get_brake_settings](#), [set_brake_settings](#)

5.3.2 Field Documentation

5.3.2.1 BrakeFlags

```
unsigned int BrakeFlags
```

[Brake settings flags.](#)

5.3.2.2 t1

```
unsigned int t1
```

Time in ms between turning on motor power and turning off the brake.

5.3.2.3 t2

unsigned int t2

Time in ms between the brake turning off and moving readiness.

All moving commands will execute after this interval.

5.3.2.4 t3

unsigned int t3

Time in ms between motor stop and the brake turning on.

5.3.2.5 t4

unsigned int t4

Time in ms between turning on the brake and turning off motor power.

5.4 calibration_settings_t Struct Reference

Calibration settings.

```
#include <ximc.h>
```

Data Fields

- float [CSS1.A](#)
Scaling factor for the analog measurements of the A winding current.
- float [CSS1.B](#)
Offset for the analog measurements of the A winding current.
- float [CSS2.A](#)
Scaling factor for the analog measurements of the B winding current.
- float [CSS2.B](#)
Offset for the analog measurements of the B winding current.
- float [FullCurrent.A](#)
Scaling factor for the analog measurements of the full current.
- float [FullCurrent.B](#)
Offset for the analog measurements of the full current.

5.4.1 Detailed Description

Calibration settings.

This structure contains calibration settings. These settings are used to convert bare ADC values to winding currents in mA and the full current in mA. Parameters are grouped into pairs, XXX_A and XXX_B, representing linear equation coefficients. The first one is the slope, the second one is the constant term. Thus, $XXX_Current[mA] = XXX_A[mA/ADC] * XXX_ADC_CODE[ADC] + XXX_B[mA]$.

See also

[get_calibration_settings](#)

[set_calibration_settings](#)

[get_calibration_settings](#), [set_calibration_settings](#)

5.4.2 Field Documentation

5.4.2.1 CSS1_A

float CSS1_A

Scaling factor for the analog measurements of the A winding current.

5.4.2.2 CSS1_B

float CSS1_B

Offset for the analog measurements of the A winding current.

5.4.2.3 CSS2_A

float CSS2_A

Scaling factor for the analog measurements of the B winding current.

5.4.2.4 CSS2_B

float CSS2_B

Offset for the analog measurements of the B winding current.

5.4.2.5 FullCurrent_A

float FullCurrent_A

Scaling factor for the analog measurements of the full current.

5.4.2.6 FullCurrent_B

float FullCurrent_B

Offset for the analog measurements of the full current.

5.5 calibration_t Struct Reference

Calibration structure

```
#include <ximc.h>
```

Data Fields

- double [A](#)
Conversion coefficient equal to the number of millimeters (or other user units) per one step.
- unsigned int **MicrostepMode**
Controller setting which is determine a step division mode

5.5.1 Detailed Description

Calibration structure

Where to find all values for calculations?

- XILab (don't forget to load the profile for your positioner. The profile should match the full name of your positioner, e.g., 8MT173-25-MEn1.cfg):
 - In XILab settings, go to the "User units" tab. Divide the second number by the first one — this is your A coefficient.
 - In the "DC motor" / "BLDC motor" / "Stepper motor" tab (depending on your motor type), find the "Encoder counts per turn" field and use it in the formula for calculating coefficient B.
- Profile file (open with any text editor; make sure it matches the full name of your positioner, e.g., 8MT173-25-MEn1.cfg):
 - Find Step_multiplier= and Unit_multiplier=. Divide the second by the first — this is your A coefficient.
 - Find Encoder_CPT= and use this value in the B coefficient formula instead of ENCODER_COUNTS_PER_TURN.

How to calculate Speed, Accel, Decel, and AntiplaySpeed in user units when using a stepper motor with encoder or DC/BLDC motor?

1. Load the correct profile in XILab for your positioner (e.g., 8MT173-25-MEn1.cfg).
 2. Enable Feedback encoder mode if not already enabled.
 3. Enter speed in the "Working speed" field in user units.
 4. In the "User units" tab, disable the "User units" flag to see RPM.
 5. Multiply the RPM value from "Working speed" by coefficient B. Example: $480 * 0.0000009375 = 0.00045$. This value for the 8MT173-25-MEn1 in Encoder mode equals 2 mm/s.
- Acceleration, deceleration, and antiplay speed are calculated the same way.

5.5.2 Field Documentation

5.5.2.1 A

double A

Conversion coefficient equal to the number of millimeters (or other user units) per one step.

Must be non-zero and positive. Used for position and movement conversion.

- For stepper motor without encoder, only one coefficient A is used. Conversion formula: $[\text{user_unit} \leftrightarrow \text{unit/steps}]$. Example: 800 steps = 1 mm, then $A = 1/800 = 0.00125$
- When using a stepper motor with encoder or DC/BLDC motors, the position is set in counts, but speed, acceleration/deceleration, and antiplay speed are set in RPM, so two coefficients are required:
 - A. Conversion for position: $[\text{user_unit/counts}]$. Example: 16000 counts = 1 mm, then $A = 1/16000 = 0.0000625$
 - B. Conversion for speed, acceleration/deceleration, and antiplay speed: $[60/\text{ENCODER_COUNTS_PER_TURN} * A]$. Example: $60 / 4000 * 0.0000625 = 0.0000009375$

5.6 `chart_data_t` Struct Reference

Additional device state.

```
#include <ximc.h>
```

Data Fields

- int [WindingVoltageA](#)
In case of a step motor, it contains the voltage across the winding A (in tens of mV); in case of a brushless motor, it contains the voltage on the first coil; in case of a DC motor, it contains the only winding current.
- int [WindingVoltageB](#)
In case of a step motor, it contains the voltage across the winding B (in tens of mV); in case of a brushless motor, it contains the voltage on the second winding; and in case of a DC motor, this field is not used.
- int [WindingVoltageC](#)
In case of a brushless motor, it contains the voltage on the third winding (in tens of mV); in the case of a step motor and a DC motor, the field is not used.
- int [WindingCurrentA](#)
In case of a step motor, it contains the current in the winding A (in mA); in case of a brushless motor, it contains the current in the winding A; and in case of a DC motor, it contains the only winding current.
- int [WindingCurrentB](#)
In case of a step motor, it contains the current in the winding B (in mA); in case of a brushless motor, it contains the current in the winding B; and in case of a DC motor, the field is not used.
- int [WindingCurrentC](#)
In case of a brushless motor, it contains the current in the winding C (in mA); in case of a step motor and a DC motor, the field is not used.
- unsigned int [Pot](#)
Analog input value, dimensionless.
- unsigned int [Joy](#)
The joystick position, dimensionless.
- int [AveragedPowerRatio](#)
The ratio of motor supplied power to nominal motor power, as a percentage.

5.6.1 Detailed Description

Additional device state.

This structure contains additional values such as winding's voltages, currents, and temperature.

See also

[get_chart_data](#)

[get_chart_data](#)

5.6.2 Field Documentation

5.6.2.1 AveragedPowerRatio

```
int AveragedPowerRatio
```

The ratio of motor supplied power to nominal motor power, as a percentage.

5.6.2.2 Joy

```
unsigned int Joy
```

The joystick position, dimensionless.

Range: 0..10000

5.6.2.3 Pot

```
unsigned int Pot
```

Analog input value, dimensionless.

Range: 0..10000

5.6.2.4 WindingCurrentA

```
int WindingCurrentA
```

In case of a step motor, it contains the current in the winding A (in mA); in case of a brushless motor, it contains the current in the winding A; and in case of a DC motor, it contains the only winding current.

5.6.2.5 WindingCurrentB

```
int WindingCurrentB
```

In case of a step motor, it contains the current in the winding B (in mA); in case of a brushless motor, it contains the current in the winding B; and in case of a DC motor, the field is not used.

5.6.2.6 WindingCurrentC

```
int WindingCurrentC
```

In case of a brushless motor, it contains the current in the winding C (in mA); in case of a step motor and a DC motor, the field is not used.

5.6.2.7 WindingVoltageA

```
int WindingVoltageA
```

In case of a step motor, it contains the voltage across the winding A (in tens of mV); in case of a brushless motor, it contains the voltage on the first coil; in case of a DC motor, it contains the only winding current.

5.6.2.8 WindingVoltageB

```
int WindingVoltageB
```

In case of a step motor, it contains the voltage across the winding B (in tens of mV); in case of a brushless motor, it contains the voltage on the second winding; and in case of a DC motor, this field is not used.

5.6.2.9 WindingVoltageC

```
int WindingVoltageC
```

In case of a brushless motor, it contains the voltage on the third winding (in tens of mV); in the case of a step motor and a DC motor, the field is not used.

5.7 control_settings_calb_t Struct Reference

User unit control settings.

```
#include <ximc.h>
```

Data Fields

- float [MaxSpeed](#) [10]
Array of speeds used with the joystick and the button control.
- unsigned int [Timeout](#) [9]
Timeout[i] is timeout in ms.
- unsigned int [MaxClickTime](#)
Maximum click time (in ms).
- unsigned int [Flags](#)
Control flags.
- float [DeltaPosition](#)
Position shift (delta)

5.7.1 Detailed Description

User unit control settings.

This structure contains control parameters.

In case of CTL_MODE=1, the joystick motor control is enabled. In this mode, while the joystick is maximally displaced, the engine tends to move at MaxSpeed[i]. i=0 if another value hasn't been set at the previous usage. To change the speed index "i", use the buttons.

In case of CTL_MODE=2, the motor is controlled by the left/right buttons. When you click on the button, the motor starts moving in the appropriate direction at a speed MaxSpeed[0]. After Timeout[i], the motor moves at speed MaxSpeed[i+1]. At the transition between MaxSpeed[i] and MaxSpeed[i+1] the motor just accelerates/decelerates as usual.

See also

[set_control_settings_calb](#)

[get_control_settings_calb](#)

[get_control_settings](#), [set_control_settings](#)

5.7.2 Field Documentation

5.7.2.1 Flags

unsigned int Flags

[Control flags](#).

5.7.2.2 MaxClickTime

unsigned int MaxClickTime

Maximum click time (in ms).

Until the expiration of this time, the first speed isn't applied.

5.7.2.3 MaxSpeed

float MaxSpeed[10]

Array of speeds used with the joystick and the button control.

5.7.2.4 Timeout

unsigned int Timeout[9]

Timeout[i] is timeout in ms.

After that, max_speed[i+1] is applied. It's used with the button control only.

5.8 control_settings_t Struct Reference

Control settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [MaxSpeed](#) [10]
Array of speeds (full step) used with the joystick and the button control.
- unsigned int [uMaxSpeed](#) [10]
Array of speeds (in microsteps) used with the joystick and the button control.
- unsigned int [Timeout](#) [9]
Timeout[i] is timeout in ms.
- unsigned int [MaxClickTime](#)
Maximum click time (in ms).
- unsigned int [Flags](#)
Control flags.
- int [DeltaPosition](#)
Position Shift (delta) (full step)
- int [uDeltaPosition](#)
Fractional part of the shift in micro steps.

5.8.1 Detailed Description

Control settings.

This structure contains control parameters.

In case of CTL_MODE=1, the joystick motor control is enabled. In this mode, while the joystick is maximally displaced, the engine tends to move at `MaxSpeed[i]`. `i=0` if another value hasn't been set at the previous usage. To change the speed index "i", use the buttons.

In case of CTL_MODE=2, the motor is controlled by the left/right buttons. When you click on the button, the motor starts moving in the appropriate direction at a speed `MaxSpeed[0]`. After `Timeout[i]`, motor moves at speed `MaxSpeed[i+1]`. At the transition between `MaxSpeed[i]` and `MaxSpeed[i+1]` the motor just accelerates/decelerates as usual.

See also

[set_control_settings](#)
[get_control_settings](#)
[get_control_settings](#), [set_control_settings](#)

5.8.2 Field Documentation

5.8.2.1 Flags

unsigned int `Flags`

[Control flags.](#)

5.8.2.2 MaxClickTime

```
unsigned int MaxClickTime
```

Maximum click time (in ms).

Until the expiration of this time, the first speed isn't applied.

5.8.2.3 MaxSpeed

```
unsigned int MaxSpeed[10]
```

Array of speeds (full step) used with the joystick and the button control.

Range: 0..100000.

5.8.2.4 Timeout

```
unsigned int Timeout[9]
```

Timeout[i] is timeout in ms.

After that, max_speed[i+1] is applied. It's used with the button control only.

5.8.2.5 uDeltaPosition

```
int uDeltaPosition
```

Fractional part of the shift in micro steps.

It's used with a stepper motor only. The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings).

5.8.2.6 uMaxSpeed

```
unsigned int uMaxSpeed[10]
```

Array of speeds (in microsteps) used with the joystick and the button control.

The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings).

5.9 controller_name_t Struct Reference

Controller name and settings flags

```
#include <ximc.h>
```

Data Fields

- char [ControllerName](#) [17]
User controller name.
- unsigned int [CtrlFlags](#)
Flags of internal controller settings.

5.9.1 Detailed Description

Controller name and settings flags

See also

[get_controller_name](#), [set_controller_name](#)

5.9.2 Field Documentation

5.9.2.1 ControllerName

```
char ControllerName[17]
```

User controller name.

It may be set by the user. Max string length: 16 characters.

5.9.2.2 CtrlFlags

```
unsigned int CtrlFlags
```

[Flags of internal controller settings.](#)

5.10 ctp_settings_t Struct Reference

Control position settings (used with stepper motor only)

```
#include <ximc.h>
```

Data Fields

- unsigned int [CTPMinError](#)
The minimum difference between the SM position in steps and the encoder position that causes the setting of the STATE_CTP_ERROR flag.
- unsigned int [CTPFlags](#)
Position control flags.

5.10.1 Detailed Description

Control position settings (used with stepper motor only)

When controlling the step motor with the encoder (CTP_BASE=0), it is possible to detect the loss of steps. The controller knows the number of steps per revolution (GENG::StepsPerRev) and the encoder resolution (GFBS::IPT). When the control is enabled (CTP_ENABLED is set), the controller stores the current position in the steps of SM and the current position of the encoder. Next, the encoder position is converted into steps at each step, and if the difference between the current position in steps and the encoder position is greater than CTPMinError, the flag STATE_CTP_ERROR is set.

Alternatively, the stepper motor may be controlled with the speed sensor (CTP_BASE 1). In this mode, at the active edges of the input clock, the controller stores the current value of steps. Then, at each revolution, the controller checks how many steps have been passed. When the difference is over the CTPMinError, the STATE_CTP_ERROR flag is set.

See also

[set_ctp_settings](#)

[get_ctp_settings](#)

[get_ctp_settings](#), [set_ctp_settings](#)

5.10.2 Field Documentation

5.10.2.1 CTPFlags

```
unsigned int CTPFlags
```

[Position control flags.](#)

5.10.2.2 CTPMinError

```
unsigned int CTPMinError
```

The minimum difference between the SM position in steps and the encoder position that causes the setting of the STATE_CTP_ERROR flag.

Measured in steps.

5.11 debug_read_t Struct Reference

Debug data.

```
#include <ximc.h>
```

Data Fields

- uint8_t [DebugData](#) [128]
Arbitrary debug data.

5.11.1 Detailed Description

Debug data.

These data are used for device debugging by the manufacturer.

See also

[get_debug_read](#)

5.11.2 Field Documentation

5.11.2.1 DebugData

```
uint8_t DebugData[128]
```

Arbitrary debug data.

5.12 debug_write_t Struct Reference

Debug data.

```
#include <ximc.h>
```

Data Fields

- uint8_t [DebugData](#) [128]
Arbitrary debug data.

5.12.1 Detailed Description

Debug data.

These data are used for device debugging by the manufacturer.

See also

[set_debug_write](#)

5.12.2 Field Documentation

5.12.2.1 DebugData

```
uint8_t DebugData[128]
```

Arbitrary debug data.

5.13 device_information_t Struct Reference

Controller information structure.

```
#include <ximc.h>
```

Data Fields

- char **Manufacturer** [5]
Manufacturer
- char **ManufacturerId** [3]
Manufacturer id
- char **ProductDescription** [9]
Product description
- unsigned int **Major**
The major number of the hardware version.
- unsigned int **Minor**
The minor number of the hardware version.
- unsigned int **Release**
Release version.

5.13.1 Detailed Description

Controller information structure.

See also

[get_device_information](#)
[get_device_information_impl](#)

5.13.2 Field Documentation

5.13.2.1 Major

```
unsigned int Major
```

The major number of the hardware version.

5.13.2.2 Minor

unsigned int Minor

The minor number of the hardware version.

5.13.2.3 Release

unsigned int Release

Release version.

5.14 device_network_information_t Struct Reference

Device network information structure.

```
#include <ximc.h>
```

Data Fields

- uint32_t **ipv4**
IPv4 address, passed in network byte order (big-endian byte order)
- char **nodename** [16]
name of the Bindy node which hosts the device
- uint32_t **axis_state**
flags representing device state
- char **locker_username** [16]
name of the user who locked the device (if any)
- char **locker_nodename** [16]
Bindy node name, which was used to lock the device (if any)
- time_t **locked_time**
time the lock was acquired at (UTC, microseconds since the epoch)

5.14.1 Detailed Description

Device network information structure.

5.15 edges_settings_calb_t Struct Reference

User unit edges settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [BorderFlags](#)
Border flags.
- unsigned int [EnderFlags](#)
Limit switches flags.
- float [LeftBorder](#)
Left border position, used if BORDER_IS_ENCODER flag is set.
- float [RightBorder](#)
Right border position, used if BORDER_IS_ENCODER flag is set.

5.15.1 Detailed Description

User unit edges settings.

This structure contains border and limit switches settings. Please load new engine settings when you change positioner, etc. Please note that wrong engine settings may lead to device malfunction, which can cause irreversible damage to the board.

See also

[set_edges_settings_calb](#)
[get_edges_settings_calb](#)
[get_edges_settings](#), [set_edges_settings](#)

5.15.2 Field Documentation

5.15.2.1 BorderFlags

unsigned int BorderFlags

[Border flags.](#)

5.15.2.2 EnderFlags

unsigned int EnderFlags

[Limit switches flags.](#)

5.15.2.3 LeftBorder

float LeftBorder

Left border position, used if BORDER_IS_ENCODER flag is set.

Corrected by the table.

5.15.2.4 RightBorder

float RightBorder

Right border position, used if BORDER_IS_ENCODER flag is set.

Corrected by the table.

5.16 edges_settings_t Struct Reference

Edges settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [BorderFlags](#)
Border flags.
- unsigned int [EnderFlags](#)
Limit switches flags.
- int [LeftBorder](#)
Left border position, used if BORDER_IS_ENCODER flag is set.
- int [uLeftBorder](#)
Left border position in microsteps (used with stepper motor only).
- int [RightBorder](#)
Right border position, used if BORDER_IS_ENCODER flag is set.
- int [uRightBorder](#)
Right border position in microsteps.

5.16.1 Detailed Description

Edges settings.

This structure contains border and limit switches settings. Please load new engine settings when you change positioner, etc. Please note that wrong engine settings may lead to device malfunction, which can cause irreversible damage to the board.

See also

[set_edges_settings](#)
[get_edges_settings](#)
[get_edges_settings](#), [set_edges_settings](#)

5.16.2 Field Documentation

5.16.2.1 BorderFlags

`unsigned int BorderFlags`

[Border flags](#).

5.16.2.2 EnderFlags

`unsigned int EnderFlags`

[Limit switches flags](#).

5.16.2.3 LeftBorder

`int LeftBorder`

Left border position, used if BORDER_IS_ENCODER flag is set.

5.16.2.4 RightBorder

`int RightBorder`

Right border position, used if BORDER_IS_ENCODER flag is set.

5.16.2.5 uLeftBorder

`int uLeftBorder`

Left border position in microsteps (used with stepper motor only).

The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings).

5.16.2.6 uRightBorder

`int uRightBorder`

Right border position in microsteps.

Used with a stepper motor only. The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings).

5.17 emf_settings_t Struct Reference

EMF settings.

```
#include <ximc.h>
```

Data Fields

- float [L](#)
Motor winding inductance.
- float [R](#)
Motor winding resistance.
- float [Km](#)
Electromechanical ratio of the motor.
- unsigned int [BackEMFFlags](#)
Flags of auto-detection of characteristics of windings of the engine.

5.17.1 Detailed Description

EMF settings.

This structure contains the data for Electromechanical characteristics (EMF) of the motor. It determines the inductance, resistance, and Electromechanical coefficient of the motor. This data is stored in the flash memory of the controller. Please set new settings when you change the motor. Remember that improper EMF settings may damage the equipment.

See also

[set_emf_settings](#)
[get_emf_settings](#)
[get_emf_settings](#), [set_emf_settings](#)

5.17.2 Field Documentation

5.17.2.1 BackEMFFlags

```
unsigned int BackEMFFlags
```

Flags of auto-detection of characteristics of windings of the engine.

5.17.2.2 Km

```
float Km
```

Electromechanical ratio of the motor.

5.17.2.3 L

float L

Motor winding inductance.

5.17.2.4 R

float R

Motor winding resistance.

5.18 encoder_information_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- char [Manufacturer](#) [17]
Manufacturer.
- char [PartNumber](#) [25]
Series and PartNumber.

5.18.1 Detailed Description

Deprecated.

Encoder information.

See also

[set_encoder_information](#)
[get_encoder_information](#)
[get_encoder_information](#), [set_encoder_information](#)

5.18.2 Field Documentation

5.18.2.1 Manufacturer

```
char Manufacturer[17]
```

Manufacturer.

Max string length: 16 chars.

5.18.2.2 PartNumber

```
char PartNumber[25]
```

Series and PartNumber.

Max string length: 24 chars.

5.19 encoder_settings_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- float [MaxOperatingFrequency](#)
Maximum operation frequency (kHz).
- float [SupplyVoltageMin](#)
Minimum supply voltage (V).
- float [SupplyVoltageMax](#)
Maximum supply voltage (V).
- float [MaxCurrentConsumption](#)
Max current consumption (mA).
- unsigned int **PPR**
The number of counts per revolution
- unsigned int [EncoderSettings](#)
Encoder settings flags.

5.19.1 Detailed Description

Deprecated.

Encoder settings.

See also

[set_encoder_settings](#)
[get_encoder_settings](#)
[get_encoder_settings](#), [set_encoder_settings](#)

5.19.2 Field Documentation

5.19.2.1 EncoderSettings

```
unsigned int EncoderSettings
```

[Encoder settings flags.](#)

5.19.2.2 MaxCurrentConsumption

float MaxCurrentConsumption

Max current consumption (mA).

Data type: float.

5.19.2.3 MaxOperatingFrequency

float MaxOperatingFrequency

Maximum operation frequency (kHz).

Data type: float.

5.19.2.4 SupplyVoltageMax

float SupplyVoltageMax

Maximum supply voltage (V).

Data type: float.

5.19.2.5 SupplyVoltageMin

float SupplyVoltageMin

Minimum supply voltage (V).

Data type: float.

5.20 engine_advanced_setup_t Struct Reference

EAS settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [stepcloseloop_Kw](#)
Manufacturer only.
- unsigned int [stepcloseloop_Kp_low](#)
Manufacturer only.
- unsigned int [stepcloseloop_Kp_high](#)
Manufacturer only.

5.20.1 Detailed Description

EAS settings.

Manufacturer only. This structure is intended for setting parameters of algorithms that cannot be attributed to standard Kp, Ki, Kd, and L, R, Km.

See also

[set_engine_advanced_setup](#)

[get_engine_advanced_setup](#)

[get_engine_advanced_setup](#), [set_engine_advanced_setup](#)

5.20.2 Field Documentation

5.20.2.1 stepcloseloop_Kp_high

```
unsigned int stepcloseloop_Kp_high
```

Manufacturer only.

Position feedback in the high-speed zone, range [0, 65535], default value 33.

5.20.2.2 stepcloseloop_Kp_low

```
unsigned int stepcloseloop_Kp_low
```

Manufacturer only.

Position feedback in the low-speed zone, range [0, 65535], default value 1000.

5.20.2.3 stepcloseloop_Kw

```
unsigned int stepcloseloop_Kw
```

Manufacturer only.

Mixing ratio of the actual and set speed, range [0, 100], default value 50.

5.21 engine_settings_calb_t Struct Reference

Movement limitations and settings, related to the motor.

```
#include <ximc.h>
```

Data Fields

- unsigned int [NomVoltage](#)
Rated voltage in tens of mV.
- unsigned int [NomCurrent](#)
Rated current (in mA).
- float [NomSpeed](#)
Nominal speed.
- unsigned int [EngineFlags](#)
Flags of engine settings.
- float [Antiplay](#)
Number of pulses or steps for backlash (play) compensation procedure.
- unsigned int [MicrostepMode](#)
Flags of microstep mode.
- unsigned int [StepsPerRev](#)
Number of full steps per revolution (Used with stepper motor only).

5.21.1 Detailed Description

Movement limitations and settings, related to the motor.

In user units.

This structure contains useful motor settings. These settings specify the motor shaft movement algorithm, list of limitations and rated characteristics. All boards are supplied with the standard set of engine settings on the controller's flash memory. Please load new engine settings when you change the motor, encoder, positioner, etc. Please note that wrong engine settings may lead to the device malfunction, that may cause irreversible damage to the board.

See also

[set_engine_settings_calb](#)
[get_engine_settings_calb](#)
[get_engine_settings](#), [set_engine_settings](#)

5.21.2 Field Documentation

5.21.2.1 Antiplay

float Antiplay

Number of pulses or steps for backlash (play) compensation procedure.

Used if ENGINE_ANTIPLAY flag is set.

5.21.2.2 EngineFlags

unsigned int EngineFlags

[Flags of engine settings.](#)

5.21.2.3 MicrostepMode

unsigned int MicrostepMode

[Flags of microstep mode.](#)

5.21.2.4 NomCurrent

unsigned int NomCurrent

Rated current (in mA).

Controller will keep current consumed by motor below this value if ENGINE_LIMIT_CURR flag is set. Range: 15..8000

5.21.2.5 NomSpeed

float NomSpeed

Nominal speed.

Controller will keep motor speed below this value if ENGINE_LIMIT_RPM flag is set.

5.21.2.6 NomVoltage

unsigned int NomVoltage

Rated voltage in tens of mV.

Controller will keep the voltage drop on motor below this value if ENGINE_LIMIT_VOLT flag is set (used with DC only).

5.21.2.7 StepsPerRev

unsigned int StepsPerRev

Number of full steps per revolution (Used with stepper motor only).

Range: 1..65535.

5.22 engine_settings_t Struct Reference

Movement limitations and settings related to the motor.

```
#include <ximc.h>
```


Data Fields

- unsigned int [NomVoltage](#)
Rated voltage in tens of mV.
- unsigned int [NomCurrent](#)
Rated current (in mA).
- unsigned int [NomSpeed](#)
Nominal (maximum) speed (in whole steps/s or rpm for DC and stepper motor as a master encoder).
- unsigned int [uNomSpeed](#)
The fractional part of a nominal speed in microsteps (is only used with stepper motor).
- unsigned int [EngineFlags](#)
Flags of engine settings.
- int [Antiplay](#)
Number of pulses or steps for backlash (play) compensation procedure.
- unsigned int [MicrostepMode](#)
Flags of microstep mode.
- unsigned int [StepsPerRev](#)
Number of full steps per revolution (Used with stepper motor only).

5.22.1 Detailed Description

Movement limitations and settings related to the motor.

This structure contains useful motor settings. These settings specify the motor shaft movement algorithm, list of limitations and rated characteristics. All boards are supplied with the standard set of engine settings on the controller's flash memory. Please load new engine settings when you change the motor, encoder, positioner, etc. Please note that wrong engine settings may lead to device malfunction, which can lead to irreversible damage to the board.

See also

[set_engine_settings](#)
[get_engine_settings](#)
[get_engine_settings](#), [set_engine_settings](#)

5.22.2 Field Documentation

5.22.2.1 Antiplay

int Antiplay

Number of pulses or steps for backlash (play) compensation procedure.

Used if ENGINE_ANTIPLAY flag is set.

5.22.2.2 EngineFlags

unsigned int EngineFlags

Flags of engine settings.

5.22.2.3 MicrostepMode

unsigned int MicrostepMode

Flags of microstep mode.

5.22.2.4 NomCurrent

unsigned int NomCurrent

Rated current (in mA).

Controller will keep current consumed by motor below this value if ENGINE_LIMIT_CURR flag is set. Range: 15..8000

5.22.2.5 NomSpeed

unsigned int NomSpeed

Nominal (maximum) speed (in whole steps/s or rpm for DC and stepper motor as a master encoder).

Controller will keep motor shaft RPM below this value if ENGINE_LIMIT_RPM flag is set. Range: 1..100000.

5.22.2.6 NomVoltage

unsigned int NomVoltage

Rated voltage in tens of mV.

Controller will keep the voltage drop on motor below this value if ENGINE_LIMIT_VOLT flag is set (used with DC only).

5.22.2.7 StepsPerRev

unsigned int StepsPerRev

Number of full steps per revolution (Used with stepper motor only).

Range: 1..65535.

5.22.2.8 uNomSpeed

unsigned int uNomSpeed

The fractional part of a nominal speed in microsteps (is only used with stepper motor).

Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).

5.23 entype_settings_t Struct Reference

Engine type and driver type settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [EngineType](#)
Flags of engine type.
- unsigned int [DriverType](#)
Flags of driver type.

5.23.1 Detailed Description

Engine type and driver type settings.

Parameters

<i>id</i>	An identifier of a device
<i>EngineType</i>	engine type
<i>DriverType</i>	driver type

See also

[get_entype_settings](#), [set_entype_settings](#)

5.23.2 Field Documentation

5.23.2.1 DriverType

unsigned int DriverType

[Flags of driver type.](#)

5.23.2.2 EngineType

`unsigned int EngineType`

Flags of engine type.

5.24 `extended_settings_t` Struct Reference

EST settings This data is stored in the controller's flash memory.

```
#include <ximc.h>
```

Data Fields

- unsigned int **Param1**

5.24.1 Detailed Description

EST settings This data is stored in the controller's flash memory.

This structure is designed for the future. Currently, it is not in use.

See also

[set_extended_settings](#)

[get_extended_settings](#)

[get_extended_settings](#), [set_extended_settings](#)

5.25 `extio_settings_t` Struct Reference

EXTIO settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [EXTIOSetupFlags](#)
External IO setup flags.
- unsigned int [EXTIOModeFlags](#)
External IO mode flags.

5.25.1 Detailed Description

EXTIO settings.

This structure contains all EXTIO settings. By default, input events are signaled through a rising front, and output states are signaled by a high logic state.

See also

[get_extio_settings](#)

[set_extio_settings](#)

[get_extio_settings](#), [set_extio_settings](#)

5.25.2 Field Documentation

5.25.2.1 EXTIOModeFlags

unsigned int EXTIOModeFlags

[External IO mode flags.](#)

5.25.2.2 EXTIOSetupFlags

unsigned int EXTIOSetupFlags

[External IO setup flags.](#)

5.26 feedback_settings_t Struct Reference

Feedback settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [IPS](#)
The number of encoder counts per shaft revolution.
- unsigned int [FeedbackType](#)
Feedback type.
- unsigned int [FeedbackFlags](#)
Describes feedback flags.
- unsigned int [CountsPerTurn](#)
The number of encoder counts per shaft revolution.

5.26.1 Detailed Description

Feedback settings.

This structure contains feedback settings.

See also

[get_feedback_settings](#), [set_feedback_settings](#)

5.26.2 Field Documentation

5.26.2.1 CountsPerTurn

```
unsigned int CountsPerTurn
```

The number of encoder counts per shaft revolution.

Range: 1..4294967295. To use the CountsPerTurn field, write 0 in the IPS field, otherwise the value from the IPS field will be used.

5.26.2.2 FeedbackFlags

```
unsigned int FeedbackFlags
```

[Describes feedback flags.](#)

5.26.2.3 FeedbackType

```
unsigned int FeedbackType
```

[Feedback type.](#)

5.26.2.4 IPS

```
unsigned int IPS
```

The number of encoder counts per shaft revolution.

Range: 1..655535. The field is obsolete, it is recommended to write 0 to IPS and use the extended CountsPerTurn field. You may need to update the controller firmware to the latest version.

5.27 gear_information_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- char [Manufacturer](#) [17]
Manufacturer.
- char [PartNumber](#) [25]
Series and PartNumber.

5.27.1 Detailed Description

Deprecated.

Gear information.

See also

[set_gear_information](#)
[get_gear_information](#)
[get_gear_information](#), [set_gear_information](#)

5.27.2 Field Documentation

5.27.2.1 Manufacturer

```
char Manufacturer[17]
```

Manufacturer.

Max string length: 16 chars.

5.27.2.2 PartNumber

```
char PartNumber[25]
```

Series and PartNumber.

Max string length: 24 chars.

5.28 gear_settings_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- float [ReductionIn](#)
Input reduction coefficient.
- float [ReductionOut](#)
Output reduction coefficient.
- float [RatedInputTorque](#)
*Maximum continuous torque ($N * m$).*
- float [RatedInputSpeed](#)
Maximum speed on the input shaft (rpm).
- float [MaxOutputBacklash](#)
Output backlash of the reduction gear (degree).
- float [InputInertia](#)
*Equivalent input gear inertia ($g * cm^2$).*
- float [Efficiency](#)
Reduction gear efficiency (%).

5.28.1 Detailed Description

Deprecated.

Gear settings.

See also

[set_gear_settings](#)

[get_gear_settings](#)

[get_gear_settings](#), [set_gear_settings](#)

5.28.2 Field Documentation

5.28.2.1 Efficiency

float Efficiency

Reduction gear efficiency (%).

Data type: float.

5.28.2.2 InputInertia

float InputInertia

Equivalent input gear inertia ($g * cm^2$).

Data type: float.

5.28.2.3 `MaxOutputBacklash`

`float MaxOutputBacklash`

Output backlash of the reduction gear (degree).

Data type: float.

5.28.2.4 `RatedInputSpeed`

`float RatedInputSpeed`

Maximum speed on the input shaft (rpm).

Data type: float.

5.28.2.5 `RatedInputTorque`

`float RatedInputTorque`

Maximum continuous torque (N * m).

Data type: float.

5.28.2.6 `ReductionIn`

`float ReductionIn`

Input reduction coefficient.

(Output = (ReductionOut / ReductionIn) * Input) Data type: float.

5.28.2.7 `ReductionOut`

`float ReductionOut`

Output reduction coefficient.

(Output = (ReductionOut / ReductionIn) * Input) Data type: float.

5.29 `get_position_calb_t` Struct Reference

Position information.

```
#include <ximc.h>
```

Data Fields

- float [Position](#)
The position in the engine.
- long_t [EncPosition](#)
Encoder position.

5.29.1 Detailed Description

Position information.

A useful structure that contains position value in user units for stepper motor and encoder steps for all engines.

See also

[get_position](#)

5.29.2 Field Documentation

5.29.2.1 EncPosition

long_t EncPosition

Encoder position.

5.29.2.2 Position

float Position

The position in the engine.

Corrected by the table.

5.30 `get_position_t` Struct Reference

Position information.

```
#include <ximc.h>
```

Data Fields

- int **Position**
The position of the whole steps in the engine
- int [uPosition](#)
Microstep position is only used with stepper motors.
- long_t [EncPosition](#)
Encoder position.

5.30.1 Detailed Description

Position information.

A useful structure that contains position value in steps and microsteps for stepper motor and encoder steps for all engines.

See also

[get_position](#)

5.30.2 Field Documentation

5.30.2.1 EncPosition

`long_t EncPosition`

Encoder position.

5.30.2.2 uPosition

`int uPosition`

Microstep position is only used with stepper motors.

Microstep size and the range of valid values for this field depend on the selected step division mode (see MicrostepMode field in engine_settings).

5.31 globally_unique_identifier_t Struct Reference

Globally unique identifier.

```
#include <ximc.h>
```

Data Fields

- unsigned int [UniqueID0](#)
Unique ID 0.
- unsigned int [UniqueID1](#)
Unique ID 1.
- unsigned int [UniqueID2](#)
Unique ID 2.
- unsigned int [UniqueID3](#)
Unique ID 3.

5.31.1 Detailed Description

Globally unique identifier.

Manufacturer only.

See also

[get_globally_unique_identifier](#)

5.31.2 Field Documentation

5.31.2.1 UniqueID0

`unsigned int UniqueID0`

Unique ID 0.

5.31.2.2 UniqueID1

`unsigned int UniqueID1`

Unique ID 1.

5.31.2.3 UniqueID2

`unsigned int UniqueID2`

Unique ID 2.

5.31.2.4 UniqueID3

`unsigned int UniqueID3`

Unique ID 3.

5.32 hallsensor_information_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- char [Manufacturer](#) [17]
Manufacturer.
- char [PartNumber](#) [25]
Series and PartNumber.

5.32.1 Detailed Description

Deprecated.

Hall sensor information.

See also

[set_hallsensor_information](#)

[get_hallsensor_information](#)

[get_hallsensor_information](#), [set_hallsensor_information](#)

5.32.2 Field Documentation

5.32.2.1 Manufacturer

```
char Manufacturer[17]
```

Manufacturer.

Max string length: 16 chars.

5.32.2.2 PartNumber

```
char PartNumber[25]
```

Series and PartNumber.

Max string length: 24 chars.

5.33 hallsensor_settings_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- float [MaxOperatingFrequency](#)
Maximum operation frequency (kHz).
- float [SupplyVoltageMin](#)
Minimum supply voltage (V).
- float [SupplyVoltageMax](#)
Maximum supply voltage (V).
- float [MaxCurrentConsumption](#)
Maximum current consumption (mA).
- unsigned int **PPR**
The number of counts per revolution

5.33.1 Detailed Description

Deprecated.

Hall sensor settings.

See also

[set_hallsensor_settings](#)

[get_hallsensor_settings](#)

[get_hallsensor_settings](#), [set_hallsensor_settings](#)

5.33.2 Field Documentation

5.33.2.1 MaxCurrentConsumption

`float MaxCurrentConsumption`

Maximum current consumption (mA).

Data type: float.

5.33.2.2 MaxOperatingFrequency

`float MaxOperatingFrequency`

Maximum operation frequency (kHz).

Data type: float.

5.33.2.3 SupplyVoltageMax

float SupplyVoltageMax

Maximum supply voltage (V).

Data type: float.

5.33.2.4 SupplyVoltageMin

float SupplyVoltageMin

Minimum supply voltage (V).

Data type: float.

5.34 home_settings_calb_t Struct Reference

Position calibration settings which use user units.

```
#include <ximc.h>
```

Data Fields

- float [FastHome](#)
Speed used for first motion.
- float [SlowHome](#)
Speed used for second motion.
- float [HomeDelta](#)
Distance from break point.
- unsigned int [HomeFlags](#)
Home settings flags.

5.34.1 Detailed Description

Position calibration settings which use user units.

This structure contains settings used in position calibrating. It specifies behavior of calibrating position.

See also

[get_home_settings_calb](#)
[set_home_settings_calb](#)
[command_home](#)
[get_home_settings](#), [set_home_settings](#)

5.34.2 Field Documentation

5.34.2.1 FastHome

float FastHome

Speed used for first motion.

5.34.2.2 HomeDelta

float HomeDelta

Distance from break point.

5.34.2.3 HomeFlags

unsigned int HomeFlags

[Home settings flags](#).

5.34.2.4 SlowHome

float SlowHome

Speed used for second motion.

5.35 home_settings_t Struct Reference

Position calibration settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [FastHome](#)
Speed used for first motion (full steps).
- unsigned int [uFastHome](#)
Fractional part of the speed for first motion, microsteps.
- unsigned int [SlowHome](#)
Speed used for second motion (full steps).
- unsigned int [uSlowHome](#)
Part of the speed for second motion, microsteps.
- int [HomeDelta](#)
Distance from break point (full steps).
- int [uHomeDelta](#)
Fractional part of the delta distance, microsteps.
- unsigned int [HomeFlags](#)
[Home settings flags](#).

5.35.1 Detailed Description

Position calibration settings.

This structure contains settings used in position calibration. It specifies behavior of calibration procedure.

See also

[get_home_settings](#)

[set_home_settings](#)

[command_home](#)

[get_home_settings](#), [set_home_settings](#)

5.35.2 Field Documentation

5.35.2.1 FastHome

`unsigned int FastHome`

Speed used for first motion (full steps).

Range: 0..100000.

5.35.2.2 HomeDelta

`int HomeDelta`

Distance from break point (full steps).

5.35.2.3 HomeFlags

`unsigned int HomeFlags`

[Home settings flags](#).

5.35.2.4 SlowHome

`unsigned int SlowHome`

Speed used for second motion (full steps).

Range: 0..100000.

5.35.2.5 `uFastHome`

```
unsigned int uFastHome
```

Fractional part of the speed for first motion, microsteps.

The microstep size and the range of valid values for this field depend on the selected step division mode (see the `MicrostepMode` field in `engine_settings`).

5.35.2.6 `uHomeDelta`

```
int uHomeDelta
```

Fractional part of the delta distance, microsteps.

The microstep size and the range of valid values for this field depend on the selected step division mode (see the `MicrostepMode` field in `engine_settings`).

5.35.2.7 `uSlowHome`

```
unsigned int uSlowHome
```

Part of the speed for second motion, microsteps.

The microstep size and the range of valid values for this field depend on the selected step division mode (see the `MicrostepMode` field in `engine_settings`).

5.36 `init_random_t` Struct Reference

Random key.

```
#include <ximc.h>
```

Data Fields

- `uint8_t key` [16]
Random key.

5.36.1 Detailed Description

Random key.

Manufacturer only.

Structure that contains a random key. It is used in the encryption of WKEY and SSER command contents.

See also

[get_init_random](#)

5.36.2 Field Documentation

5.36.2.1 key

```
uint8_t key[16]
```

Random key.

5.37 joystick_settings_t Struct Reference

Joystick settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [JoyLowEnd](#)
Joystick lower end position.
- unsigned int [JoyCenter](#)
Joystick center position.
- unsigned int [JoyHighEnd](#)
Joystick upper end position.
- unsigned int [ExpFactor](#)
Exponential nonlinearity factor.
- unsigned int [DeadZone](#)
Joystick deviation from the central position in 0.1% units that do not result in a start of motion.
- unsigned int [JoyFlags](#)
Joystick flags.

5.37.1 Detailed Description

Joystick settings.

This structure contains joystick parameters. If joystick position falls outside DeadZone limits, a movement begins. The speed is defined by the joystick's position in the range from the DeadZone limit to the maximum deviation. Joystick positions inside DeadZone limits correspond to zero speed (a "soft stop" command is issued continuously), and positions beyond Low and High limits correspond to MaxSpeed[i] or -MaxSpeed[i] (see command SCTL), where i = 0 by default and can be changed with the left/right buttons (see command SCTL). If the next speed in the list is zero (both integer and microstep parts), the button press is ignored. The first speed in the list shouldn't be zero. DeadZone is defined in 0.1% units.

The relationship between the deviation and the rate is exponential, which allows for high mobility and accuracy without speed mode switching.

See also

[set_joystick_settings](#)
[get_joystick_settings](#)
[get_joystick_settings](#), [set_joystick_settings](#)

5.37.2 Field Documentation

5.37.2.1 DeadZone

`unsigned int DeadZone`

Joystick deviation from the central position in 0.1% units that do not result in a start of motion.

Maximal deviation is $\pm 25.5\%$ that is more than a half of the total joystick range.

5.37.2.2 ExpFactor

`unsigned int ExpFactor`

Exponential nonlinearity factor.

5.37.2.3 JoyCenter

`unsigned int JoyCenter`

Joystick center position.

Range: 0..10000.

5.37.2.4 JoyFlags

`unsigned int JoyFlags`

[Joystick flags](#).

5.37.2.5 JoyHighEnd

`unsigned int JoyHighEnd`

Joystick upper end position.

Range: 0..10000.

5.37.2.6 JoyLowEnd

`unsigned int JoyLowEnd`

Joystick lower end position.

Range: 0..10000.

5.38 measurements_t Struct Reference

The structure contains a sequence of measured axis motion parameters - velocities and position errors.

```
#include <ximc.h>
```

Data Fields

- int **Speed** [25]
Sequence of measured speeds (in encoder counts/s or microsteps/sec, depending on the motor type and control mode)
- int **Error** [25]
Position error in microsteps (whole steps are recalculated considering the current step division mode) or encoder counts.
- unsigned int **Length**
Actual sequence length.

5.38.1 Detailed Description

The structure contains a sequence of measured axis motion parameters - velocities and position errors.

The time step between consecutive measurements is 1 ms.

See also

[measurements](#)
[get_measurements](#)

5.38.2 Field Documentation

5.38.2.1 Error

```
int Error[25]
```

Position error in microsteps (whole steps are recalculated considering the current step division mode) or encoder counts.

5.38.2.2 Length

```
unsigned int Length
```

Actual sequence length.

The values contained in cells Length, Length+1, ...24 shall not be interpreted as measurement results.

5.39 motor_information_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- char [Manufacturer](#) [17]
Manufacturer.
- char [PartNumber](#) [25]
Series and PartNumber.

5.39.1 Detailed Description

Deprecated.

motor information.

See also

[set_motor_information](#)
[get_motor_information](#)
[get_motor_information](#), [set_motor_information](#)

5.39.2 Field Documentation

5.39.2.1 Manufacturer

```
char Manufacturer[17]
```

Manufacturer.

Max string length: 16 chars.

5.39.2.2 PartNumber

```
char PartNumber[25]
```

Series and PartNumber.

Max string length: 24 chars.

5.40 motor_settings_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- unsigned int [MotorType](#)
Motor Type flags.
- unsigned int [ReservedField](#)
Reserved
- unsigned int [Poles](#)
Number of pole pairs for DC or BLDC motors or number of steps per rotation for stepper motors.
- unsigned int [Phases](#)
Number of phases for BLDC motors.
- float [NominalVoltage](#)
Nominal voltage on winding (B).
- float [NominalCurrent](#)
Maximum direct current in winding for DC and BLDC engines, nominal current in windings for stepper motors (A).
- float [NominalSpeed](#)
Not used.
- float [NominalTorque](#)
*Nominal torque ($mN * m$).*
- float [NominalPower](#)
Nominal power (W).
- float [WindingResistance](#)
Resistance of windings for DC engines, of each of two windings for stepper motors, or of each of three windings for BLDC engines (Ohm).
- float [WindingInductance](#)
Inductance of windings for DC engines, inductance of each of two windings for stepper motors, or inductance of each of three windings for BLDC engines (mH).
- float [RotorInertia](#)
*Rotor inertia ($g * cm^2$).*
- float [StallTorque](#)
*Torque hold position for a stepper motor or torque at a motionless rotor for other types of engines ($mN * m$).*
- float [DetentTorque](#)
*Holding torque position with unpowered windings ($mN * m$).*
- float [TorqueConstant](#)
*Torque constant that determines the proportionality constant between the maximum rotor torque and current flowing in the winding ($mN * m / A$).*
- float [SpeedConstant](#)
Velocity constant, which determines the value or the amplitude of the induced voltage on the motion of DC or BLDC motors (rpm / V) or stepper motors ($steps/s / V$).
- float [SpeedTorqueGradient](#)
*Speed torque gradient ($rpm / mN * m$).*
- float [MechanicalTimeConstant](#)
Mechanical time constant (ms).

- float [MaxSpeed](#)
The maximum speed for stepper motors (steps/s) or DC and BLDC motors (rpm).
- float [MaxCurrent](#)
The maximum current in the winding (A).
- float [MaxCurrentTime](#)
Safe duration of overcurrent in the winding (ms).
- float [NoLoadCurrent](#)
The current consumption in idle mode (A).
- float [NoLoadSpeed](#)
Idle speed (rpm).

5.40.1 Detailed Description

Deprecated.

Physical characteristics and limitations of the motor.

See also

[set_motor_settings](#)
[get_motor_settings](#)
[get_motor_settings](#), [set_motor_settings](#)

5.40.2 Field Documentation

5.40.2.1 DetentTorque

float DetentTorque

Holding torque position with unpowered windings (mN * m).

Data type: float.

5.40.2.2 MaxCurrent

float MaxCurrent

The maximum current in the winding (A).

Data type: float.

5.40.2.3 MaxCurrentTime

float MaxCurrentTime

Safe duration of overcurrent in the winding (ms).

Data type: float.

5.40.2.4 MaxSpeed

`float MaxSpeed`

The maximum speed for stepper motors (steps/s) or DC and BLDC motors (rpm).

Data type: float.

5.40.2.5 MechanicalTimeConstant

`float MechanicalTimeConstant`

Mechanical time constant (ms).

Data type: float.

5.40.2.6 MotorType

`unsigned int MotorType`

[Motor Type flags](#).

5.40.2.7 NoLoadCurrent

`float NoLoadCurrent`

The current consumption in idle mode (A).

Used for DC and BLDC motors. Data type: float.

5.40.2.8 NoLoadSpeed

`float NoLoadSpeed`

Idle speed (rpm).

Used for DC and BLDC motors. Data type: float.

5.40.2.9 NominalCurrent

`float NominalCurrent`

Maximum direct current in winding for DC and BLDC engines, nominal current in windings for stepper motors (A).

Data type: float.

5.40.2.10 NominalPower

`float NominalPower`

Nominal power (W).

Used for DC and BLDC engines. Data type: float.

5.40.2.11 NominalSpeed

`float NominalSpeed`

Not used.

Nominal speed (rpm). Used for DC and BLDC engines. Data type: float.

5.40.2.12 NominalTorque

`float NominalTorque`

Nominal torque (mN * m).

Used for DC and BLDC engines. Data type: float.

5.40.2.13 NominalVoltage

`float NominalVoltage`

Nominal voltage on winding (V).

Data type: float

5.40.2.14 Phases

`unsigned int Phases`

Number of phases for BLDC motors.

5.40.2.15 Poles

`unsigned int Poles`

Number of pole pairs for DC or BLDC motors or number of steps per rotation for stepper motors.

5.40.2.16 RotorInertia

float RotorInertia

Rotor inertia ($\text{g} * \text{cm}^2$).

Data type: float.

5.40.2.17 SpeedConstant

float SpeedConstant

Velocity constant, which determines the value or the amplitude of the induced voltage on the motion of DC or BLDC motors (rpm / V) or stepper motors ($\text{steps/s} / \text{V}$).

Data type: float.

5.40.2.18 SpeedTorqueGradient

float SpeedTorqueGradient

Speed torque gradient ($\text{rpm} / \text{mN} * \text{m}$).

Data type: float.

5.40.2.19 StallTorque

float StallTorque

Torque hold position for a stepper motor or torque at a motionless rotor for other types of engines ($\text{mN} * \text{m}$).

Data type: float.

5.40.2.20 TorqueConstant

float TorqueConstant

Torque constant that determines the proportionality constant between the maximum rotor torque and current flowing in the winding ($\text{mN} * \text{m} / \text{A}$).

Used mainly for DC motors. Data type: float.

5.40.2.21 WindingInductance

float WindingInductance

Inductance of windings for DC engines, inductance of each of two windings for stepper motors, or inductance of each of three windings for BLDC engines (mH).

Data type: float.

5.40.2.22 WindingResistance

float WindingResistance

Resistance of windings for DC engines, of each of two windings for stepper motors, or of each of three windings for BLDC engines (Ohm).

Data type: float.

5.41 move_settings_calb_t Struct Reference

User units move settings.

```
#include <ximc.h>
```

Data Fields

- float [Speed](#)
Speed
- float [Accel](#)
Acceleration:
- float [Decel](#)
Deceleration
- float [AntiplaySpeed](#)
Antiplay speed
- unsigned int [MoveFlags](#)
Flags of the motion parameters.

5.41.1 Detailed Description

User units move settings.

See also

[set_move_settings_calb](#)
[get_move_settings_calb](#)
[get_move_settings](#), [set_move_settings](#)

5.41.2 Field Documentation

5.41.2.1 Accel

float Accel

Acceleration:

- For stepper motor without encoder — in steps per second².
- When using encoder (including DC/BLDC) — in revolutions per minute (RPM).

5.41.2.2 AntiplaySpeed

float AntiplaySpeed

Antiplay speed

- For stepper motor without encoder — in steps per second.
- When using encoder (including DC/BLDC) — in revolutions per minute (RPM).

5.41.2.3 Decel

float Decel

Deceleration

- For stepper motor without encoder — in steps per second².
- When using encoder (including DC/BLDC) — in revolutions per minute (RPM).

5.41.2.4 MoveFlags

unsigned int MoveFlags

[Flags of the motion parameters.](#)

5.41.2.5 Speed

float Speed

Speed

- For stepper motor without encoder — in steps per second.
- When using encoder (including DC/BLDC) — in revolutions per minute (RPM).

5.42 move_settings_t Struct Reference

Move settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [Speed](#)
Target speed (for stepper motor: steps/s, for DC: rpm).
- unsigned int [uSpeed](#)
Target speed in microstep fractions/s.
- unsigned int [Accel](#)
Motor shaft acceleration, steps/s² (stepper motor) or RPM/s (DC).
- unsigned int [Decel](#)
Motor shaft deceleration, steps/s² (stepper motor) or RPM/s (DC).
- unsigned int [AntiplaySpeed](#)
Speed in antiplay mode, full steps/s (stepper motor) or RPM (DC).
- unsigned int [uAntiplaySpeed](#)
Speed in antiplay mode, microsteps/s.
- unsigned int [MoveFlags](#)
Flags of the motion parameters.

5.42.1 Detailed Description

Move settings.

See also

[set_move_settings](#)
[get_move_settings](#)
[get_move_settings](#), [set_move_settings](#)

5.42.2 Field Documentation

5.42.2.1 Accel

unsigned int Accel

Motor shaft acceleration, steps/s² (stepper motor) or RPM/s (DC).

Range: 1..65535.

5.42.2.2 AntiplaySpeed

unsigned int AntiplaySpeed

Speed in antiplay mode, full steps/s (stepper motor) or RPM (DC).

Range: 0..100000.

5.42.2.3 Decel

unsigned int Decel

Motor shaft deceleration, steps/s² (stepper motor) or RPM/s (DC).

Range: 1..65535.

5.42.2.4 MoveFlags

unsigned int MoveFlags

[Flags of the motion parameters.](#)

5.42.2.5 Speed

unsigned int Speed

Target speed (for stepper motor: steps/s, for DC: rpm).

Range: 0..100000.

5.42.2.6 uAntiplaySpeed

unsigned int uAntiplaySpeed

Speed in antiplay mode, microsteps/s.

The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings). Used with a stepper motor only.

5.42.2.7 uSpeed

unsigned int uSpeed

Target speed in microstep fractions/s.

The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings). Used with a stepper motor only.

5.43 network_settings_t Struct Reference

Network settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [DHCPEnabled](#)
Indicates the method to get the IP-address.
- unsigned int [IPv4Address](#) [4]
IP-address of the device in format x.x.x.x.
- unsigned int [SubnetMask](#) [4]
The mask of the subnet in format x.x.x.x.
- unsigned int [DefaultGateway](#) [4]
Default value of the gateway in format x.x.x.x.

5.43.1 Detailed Description

Network settings.

Manufacturer only. This structure contains network settings.

See also

[get_network_settings](#)

[set_network_settings](#)

[get_network_settings](#), [set_network_settings](#)

5.43.2 Field Documentation

5.43.2.1 DefaultGateway

```
unsigned int DefaultGateway[4]
```

Default value of the gateway in format x.x.x.x.

5.43.2.2 DHCPEnabled

```
unsigned int DHCPEnabled
```

Indicates the method to get the IP-address.

It can be either 0 (static) or 1 (DHCP).

5.43.2.3 IPv4Address

```
unsigned int IPv4Address[4]
```

IP-address of the device in format x.x.x.x.

5.43.2.4 SubnetMask

```
unsigned int SubnetMask[4]
```

The mask of the subnet in format x.x.x.x.

5.44 `nonvolatile_memory_t` Struct Reference

Structure contains user data to save into the FRAM.

```
#include <ximc.h>
```

Data Fields

- unsigned int [UserData](#) [7]
User data.

5.44.1 Detailed Description

Structure contains user data to save into the FRAM.

See also

[get_nonvolatile_memory](#), [set_nonvolatile_memory](#)

5.44.2 Field Documentation

5.44.2.1 UserData

```
unsigned int UserData[7]
```

User data.

It may be set by the user. Each element of the array stores only 32 bits of user data. This is important on systems where an int type contains more than 4 bytes. For example, on all amd64 systems.

5.45 `password_settings_t` Struct Reference

The web-page user password.

```
#include <ximc.h>
```

Data Fields

- char [UserPassword](#) [21]

Password for the web-page that the user can change with a USB command or via web-page.

5.45.1 Detailed Description

The web-page user password.

Manufacturer only. This structure contains the user password.

See also

[get_password_settings](#)

[set_password_settings](#)

[get_password_settings](#), [set_password_settings](#)

5.45.2 Field Documentation

5.45.2.1 UserPassword

```
char UserPassword[21]
```

Password for the web-page that the user can change with a USB command or via web-page.

5.46 pid_settings_t Struct Reference

PID settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int **KpU**
Proportional gain for voltage PID routine
- unsigned int **KiU**
Integral gain for voltage PID routine
- unsigned int **KdU**
Differential gain for voltage PID routine
- float **Kpf**
Proportional gain for BLDC position PID routine
- float **Kif**
Integral gain for BLDC position PID routine
- float **Kdf**
Differential gain for BLDC position PID routine

5.46.1 Detailed Description

PID settings.

This structure contains factors for PID routine. It specifies the behavior of the voltage PID routine. These factors are slightly different for different positioners. All boards are supplied with the standard set of PID settings in the controller's flash memory. Please load new PID settings when you change positioner. Please note that wrong PID settings lead to device malfunction.

See also

[set_pid_settings](#)

[get_pid_settings](#)

[get_pid_settings](#), [set_pid_settings](#)

5.47 `power_settings_t` Struct Reference

Step motor power settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [HoldCurrent](#)
Holding current, as percent of the nominal current.
- unsigned int [CurrReductDelay](#)
Time in ms from going to STOP state to the end of current reduction.
- unsigned int [PowerOffDelay](#)
Time in s from going to STOP state to turning power off.
- unsigned int [CurrentSetTime](#)
Time in ms to reach the nominal current.
- unsigned int [PowerFlags](#)
Flags of power settings of stepper motor.

5.47.1 Detailed Description

Step motor power settings.

See also

[set_move_settings](#)

[get_move_settings](#)

[get_power_settings](#), [set_power_settings](#)

5.47.2 Field Documentation

5.47.2.1 `CurrentSetTime`

```
unsigned int CurrentSetTime
```

Time in ms to reach the nominal current.

5.47.2.2 `CurrReductDelay`

```
unsigned int CurrReductDelay
```

Time in ms from going to STOP state to the end of current reduction.

5.47.2.3 `HoldCurrent`

```
unsigned int HoldCurrent
```

Holding current, as percent of the nominal current.

Range: 0..100.

5.47.2.4 `PowerFlags`

```
unsigned int PowerFlags
```

Flags of power settings of stepper motor.

5.47.2.5 `PowerOffDelay`

```
unsigned int PowerOffDelay
```

Time in s from going to STOP state to turning power off.

5.48 `secure_settings_t` Struct Reference

This structure contains protection parameters: critical electrical values, flags for protection algorithms.

```
#include <ximc.h>
```

Data Fields

- unsigned int [LowUpwrOff](#)
Lower voltage limit to turn off the motor, in tens of mV.
- unsigned int [CriticalIpwr](#)
Maximum motor current which triggers ALARM state, in mA.
- unsigned int [CriticalUpwr](#)
Maximum motor voltage which triggers ALARM state, in tens of mV.
- unsigned int [CriticalT](#)
Maximum temperature, which triggers ALARM state, in tenths of degrees Celsius.
- unsigned int [CriticalIusb](#)
Deprecated.
- unsigned int [CriticalUusb](#)
Deprecated.
- unsigned int [MinimumUusb](#)
Deprecated.
- unsigned int [Flags](#)
Flags of secure settings.

5.48.1 Detailed Description

This structure contains protection parameters: critical electrical values, flags for protection algorithms.

See also

[get_secure_settings](#)

[set_secure_settings](#)

[get_secure_settings](#), [set_secure_settings](#)

5.48.2 Field Documentation

5.48.2.1 `CriticalIpwr`

`unsigned int CriticalIpwr`

Maximum motor current which triggers ALARM state, in mA.

5.48.2.2 `CriticalIusb`

`unsigned int CriticalIusb`

Deprecated.

Maximum USB current which triggers ALARM state, in mA.

5.48.2.3 CriticalT

`unsigned int CriticalT`

Maximum temperature, which triggers ALARM state, in tenths of degrees Celsius.

5.48.2.4 CriticalUpwr

`unsigned int CriticalUpwr`

Maximum motor voltage which triggers ALARM state, in tens of mV.

5.48.2.5 CriticalUusb

`unsigned int CriticalUusb`

Deprecated.

Maximum USB voltage which triggers ALARM state, in tens of mV.

5.48.2.6 Flags

`unsigned int Flags`

[Flags of secure settings.](#)

5.48.2.7 LowUpwrOff

`unsigned int LowUpwrOff`

Lower voltage limit to turn off the motor, in tens of mV.

5.48.2.8 MinimumUusb

`unsigned int MinimumUusb`

Deprecated.

Minimum USB voltage which triggers ALARM state, in tens of mV.

5.49 serial_number_t Struct Reference

The structure contains a new serial number, hardware version, and valid key.

```
#include <ximc.h>
```

Data Fields

- unsigned int [SN](#)
New board serial number.
- uint8_t [Key](#) [32]
Protection key (256 bit).
- unsigned int [Major](#)
The major number of the hardware version.
- unsigned int [Minor](#)
The minor number of the hardware version.
- unsigned int [Release](#)
Number of edits this release of hardware.

5.49.1 Detailed Description

The structure contains a new serial number, hardware version, and valid key.

The SN and hardware version are changed and saved when the transmitted key matches the stored key. It can be used by the manufacturer only. Used from the loader only.

See also

[set_serial_number](#)

5.49.2 Field Documentation

5.49.2.1 Key

```
uint8_t Key[32]
```

Protection key (256 bit).

5.49.2.2 Major

```
unsigned int Major
```

The major number of the hardware version.

5.49.2.3 Minor

```
unsigned int Minor
```

The minor number of the hardware version.

5.49.2.4 Release

`unsigned int Release`

Number of edits this release of hardware.

5.49.2.5 SN

`unsigned int SN`

New board serial number.

5.50 `set_position_calb_t` Struct Reference

User unit position information.

```
#include <ximc.h>
```

Data Fields

- float [Position](#)
The position in the engine.
- long_t [EncPosition](#)
Encoder position.
- unsigned int [PosFlags](#)
Position setting flags.

5.50.1 Detailed Description

User unit position information.

A useful structure that contains position value in steps and microsteps for stepper motor and encoder steps of all engines.

See also

[set_position](#)

5.50.2 Field Documentation

5.50.2.1 EncPosition

`long_t EncPosition`

Encoder position.

5.50.2.2 PosFlags

`unsigned int PosFlags`

Position setting flags.

5.50.2.3 Position

`float Position`

The position in the engine.

5.51 `set_position_t` Struct Reference

Position information.

```
#include <ximc.h>
```

Data Fields

- `int` **Position**
The position of the whole steps in the engine
- `int` **uPosition**
Microstep position is only used with stepper motors.
- `long_t` **EncPosition**
Encoder position.
- `unsigned int` **PosFlags**
Position setting flags.

5.51.1 Detailed Description

Position information.

A useful structure that contains position value in steps and microsteps for stepper motor and encoder steps for all engines.

See also

[set_position](#)

5.51.2 Field Documentation

5.51.2.1 EncPosition

`long_t EncPosition`

Encoder position.

5.51.2.2 PosFlags

```
unsigned int PosFlags
```

Position setting flags.

5.51.2.3 uPosition

```
int uPosition
```

Microstep position is only used with stepper motors.

Microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings).

5.52 stage_information_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- char [Manufacturer](#) [17]
Manufacturer.
- char [PartNumber](#) [25]
Series and PartNumber.

5.52.1 Detailed Description

Deprecated.

Stage information. Deprecated.

See also

[set_stage_information](#)
[get_stage_information](#)
[get_stage_information](#), [set_stage_information](#)

5.52.2 Field Documentation

5.52.2.1 Manufacturer

```
char Manufacturer[17]
```

Manufacturer.

Max string length: 16 chars.

5.52.2.2 PartNumber

```
char PartNumber[25]
```

Series and PartNumber.

Max string length: 24 chars.

5.53 stage_name_t Struct Reference

Stage username.

```
#include <ximc.h>
```

Data Fields

- char [PositionerName](#) [17]
User's positioner name.

5.53.1 Detailed Description

Stage username.

See also

[get_stage_name](#), [set_stage_name](#)

5.53.2 Field Documentation

5.53.2.1 PositionerName

```
char PositionerName[17]
```

User's positioner name.

It can be set by a user. Max string length: 16 characters.

5.54 stage_settings_t Struct Reference

Deprecated.

```
#include <ximc.h>
```

Data Fields

- float [LeadScrewPitch](#)
Lead screw pitch (mm).
- char [Units](#) [9]
Units for MaxSpeed and TravelRange fields of the structure (steps, degrees, mm, ...).
- float [MaxSpeed](#)
Maximum speed (Units/c).
- float [TravelRange](#)
Travel range (Units).
- float [SupplyVoltageMin](#)
Minimum supply voltage (V).
- float [SupplyVoltageMax](#)
Maximum supply voltage (V).
- float [MaxCurrentConsumption](#)
Maximum current consumption (A).
- float [HorizontalLoadCapacity](#)
Horizontal load capacity (kg).
- float [VerticalLoadCapacity](#)
Vertical load capacity (kg).

5.54.1 Detailed Description

Deprecated.

Stage settings.

See also

[set_stage_settings](#)

[get_stage_settings](#)

[get_stage_settings](#), [set_stage_settings](#)

5.54.2 Field Documentation

5.54.2.1 HorizontalLoadCapacity

float HorizontalLoadCapacity

Horizontal load capacity (kg).

Data type: float.

5.54.2.2 LeadScrewPitch

float LeadScrewPitch

Lead screw pitch (mm).

Data type: float.

5.54.2.3 MaxCurrentConsumption

float MaxCurrentConsumption

Maximum current consumption (A).

Data type: float.

5.54.2.4 MaxSpeed

float MaxSpeed

Maximum speed (Units/c).

Data type: float.

5.54.2.5 SupplyVoltageMax

float SupplyVoltageMax

Maximum supply voltage (V).

Data type: float.

5.54.2.6 SupplyVoltageMin

float SupplyVoltageMin

Minimum supply voltage (V).

Data type: float.

5.54.2.7 TravelRange

float TravelRange

Travel range (Units).

Data type: float.

5.54.2.8 Units

char Units[9]

Units for MaxSpeed and TravelRange fields of the structure (steps, degrees, mm, ...).

Max string length: 8 chars.

5.54.2.9 VerticalLoadCapacity

```
float VerticalLoadCapacity
```

Vertical load capacity (kg).

Data type: float.

5.55 status_calb_t Struct Reference

User unit device's state.

```
#include <ximc.h>
```

Data Fields

- unsigned int [MoveSts](#)
Flags of move state.
- unsigned int [MvCmdSts](#)
Move command state.
- unsigned int [PWRSts](#)
Flags of power state of stepper motor.
- unsigned int [EncSts](#)
Encoder state.
- unsigned int [WindSts](#)
Winding state.
- float [CurPosition](#)
Current position.
- long_t [EncPosition](#)
Current encoder position.
- float [CurSpeed](#)
Motor shaft speed.
- int [lpwr](#)
Engine current, mA.
- int [Upwr](#)
Power supply voltage, tens of mV.
- int [lusb](#)
USB current, mA.
- int [Uusb](#)
USB voltage, tens of mV.
- int [CurT](#)
Temperature, tenths of degrees Celsius.
- unsigned int [Flags](#)
Status flags.
- unsigned int [GPIOFlags](#)
Status flags of the GPIO outputs.
- unsigned int [CmdBufFreeSpace](#)
This field is a service field.

5.55.1 Detailed Description

User unit device's state.

A useful structure that contains current controller state, including speed, position, and boolean flags.

See also

`get_status_impl`

5.55.2 Field Documentation

5.55.2.1 CmdBufFreeSpace

`unsigned int CmdBufFreeSpace`

This field is a service field.

It shows the number of free synchronization chain buffer cells.

5.55.2.2 CurPosition

`float CurPosition`

Current position.

Corrected by the table.

5.55.2.3 CurSpeed

`float CurSpeed`

Motor shaft speed.

5.55.2.4 CurT

`int CurT`

Temperature, tenths of degrees Celsius.

5.55.2.5 EncPosition

`long_t EncPosition`

Current encoder position.

5.55.2.6 EncSts

unsigned int EncSts

[Encoder state.](#)

5.55.2.7 Flags

unsigned int Flags

[Status flags.](#)

5.55.2.8 GPIOFlags

unsigned int GPIOFlags

[Status flags of the GPIO outputs.](#)

5.55.2.9 Ipwr

int Ipwr

Engine current, mA.

5.55.2.10 Iusb

int Iusb

USB current, mA.

5.55.2.11 MoveSts

unsigned int MoveSts

[Flags of move state.](#)

5.55.2.12 MvCmdSts

unsigned int MvCmdSts

[Move command state.](#)

5.55.2.13 PWRSts

unsigned int PWRSts

Flags of power state of stepper motor.

5.55.2.14 Upwr

int Upwr

Power supply voltage, tens of mV.

5.55.2.15 Uusb

int Uusb

USB voltage, tens of mV.

5.55.2.16 WindSts

unsigned int WindSts

Winding state.

5.56 status_t Struct Reference

Device state.

```
#include <ximc.h>
```

Data Fields

- unsigned int [MoveSts](#)
Flags of move state.
- unsigned int [MvCmdSts](#)
Move command state.
- unsigned int [PWRSts](#)
Flags of power state of stepper motor.
- unsigned int [EncSts](#)
Encoder state.
- unsigned int [WindSts](#)
Winding state.
- int [CurPosition](#)
Current position.
- int [uCurPosition](#)
Step motor shaft position in microsteps.

- `long_t` [EncPosition](#)
Current encoder position.
- `int` [CurSpeed](#)
Motor shaft speed in steps/s or rpm.
- `int` [uCurSpeed](#)
Fractional part of motor shaft speed in microsteps.
- `int` [lpwr](#)
Engine current, mA.
- `int` [Upwr](#)
Power supply voltage, tens of mV.
- `int` [lusb](#)
USB current, mA.
- `int` [Uusb](#)
USB voltage, tens of mV.
- `int` [CurT](#)
Temperature, tenths of degrees Celsius.
- `unsigned int` [Flags](#)
Status flags.
- `unsigned int` [GPIOFlags](#)
Status flags of the GPIO outputs.
- `unsigned int` [CmdBufFreeSpace](#)
This field is a service field.

5.56.1 Detailed Description

Device state.

A useful structure that contains current controller state, including speed, position, and boolean flags.

See also

`get_status_impl`

5.56.2 Field Documentation

5.56.2.1 `CmdBufFreeSpace`

```
unsigned int CmdBufFreeSpace
```

This field is a service field.

It shows the number of free synchronization chain buffer cells.

5.56.2.2 `CurPosition`

```
int CurPosition
```

Current position.

5.56.2.3 CurSpeed

```
int CurSpeed
```

Motor shaft speed in steps/s or rpm.

5.56.2.4 CurT

```
int CurT
```

Temperature, tenths of degrees Celsius.

5.56.2.5 EncPosition

```
long_t EncPosition
```

Current encoder position.

5.56.2.6 EncSts

```
unsigned int EncSts
```

[Encoder state.](#)

5.56.2.7 Flags

```
unsigned int Flags
```

[Status flags.](#)

5.56.2.8 GPIOFlags

```
unsigned int GPIOFlags
```

[Status flags of the GPIO outputs.](#)

5.56.2.9 Ipwr

```
int Ipwr
```

Engine current, mA.

5.56.2.10 Iusb

int Iusb

USB current, mA.

5.56.2.11 MoveSts

unsigned int MoveSts

Flags of move state.

5.56.2.12 MvCmdSts

unsigned int MvCmdSts

Move command state.

5.56.2.13 PWRSts

unsigned int PWRSts

Flags of power state of stepper motor.

5.56.2.14 uCurPosition

int uCurPosition

Step motor shaft position in microsteps.

The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings). Used with stepper motors only.

5.56.2.15 uCurSpeed

int uCurSpeed

Fractional part of motor shaft speed in microsteps.

The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings). Used with stepper motors only.

5.56.2.16 Upwr

int Upwr

Power supply voltage, tens of mV.

5.56.2.17 Uusb

int Uusb

USB voltage, tens of mV.

5.56.2.18 WindSts

unsigned int WindSts

[Winding state](#).

5.57 sync_in_settings_calb_t Struct Reference

User unit synchronization settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [SyncInFlags](#)
Flags for synchronization input setup.
- unsigned int [ClutterTime](#)
Input synchronization pulse dead time (us).
- float [Position](#)
Desired position or shift.
- float [Speed](#)
Target speed.
- unsigned int **reserved0**

5.57.1 Detailed Description

User unit synchronization settings.

This structure contains all synchronization settings, modes, periods and flags. It specifies behavior of input synchronization. All boards are supplied with the standard set of these settings.

See also

[get_sync_in_settings_calb](#)

[set_sync_in_settings_calb](#)

[get_sync_in_settings](#), [set_sync_in_settings](#)

5.57.2 Field Documentation

5.57.2.1 ClutterTime

unsigned int ClutterTime

Input synchronization pulse dead time (us).

5.57.2.2 Position

float Position

Desired position or shift.

5.57.2.3 Speed

float Speed

Target speed.

5.57.2.4 SyncInFlags

unsigned int SyncInFlags

Flags for synchronization input setup.

5.58 sync_in_settings_t Struct Reference

Synchronization settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [SyncInFlags](#)
Flags for synchronization input setup.
- unsigned int [ClutterTime](#)
Input synchronization pulse dead time (us).
- int **Position**
Desired position or shift (full steps)
- int [uPosition](#)
The fractional part of a position or shift in microsteps.
- unsigned int [Speed](#)
Target speed (for stepper motor: steps/s, for DC: rpm).
- unsigned int [uSpeed](#)
Target speed in microsteps/s.
- unsigned int **reserved0**

5.58.1 Detailed Description

Synchronization settings.

This structure contains all synchronization settings, modes, periods and flags. It specifies the behavior of the input synchronization. All boards are supplied with the standard set of these settings.

See also

[get_sync_in_settings](#)

[set_sync_in_settings](#)

[get_sync_in_settings](#), [set_sync_in_settings](#)

5.58.2 Field Documentation

5.58.2.1 ClutterTime

`unsigned int ClutterTime`

Input synchronization pulse dead time (us).

5.58.2.2 Speed

`unsigned int Speed`

Target speed (for stepper motor: steps/s, for DC: rpm).

Range: 0..100000.

5.58.2.3 SyncInFlags

`unsigned int SyncInFlags`

[Flags for synchronization input setup.](#)

5.58.2.4 uPosition

`int uPosition`

The fractional part of a position or shift in microsteps.

It is used with a stepper motor. The microstep size and the range of valid values for this field depend on the selected step division mode (see the `MicrostepMode` field in `engine_settings`).

5.58.2.5 uSpeed

unsigned int uSpeed

Target speed in microsteps/s.

Microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings). Used a stepper motor only.

5.59 sync_out_settings_calb_t Struct Reference

Synchronization settings which use user units.

```
#include <ximc.h>
```

Data Fields

- unsigned int [SyncOutFlags](#)
Flags of synchronization output.
- unsigned int [SyncOutPulseSteps](#)
This value specifies the duration of output pulse.
- unsigned int [SyncOutPeriod](#)
This value specifies the number of encoder pulses or steps between two output synchronization pulses when SYNCOUT_ONPERIOD is set.
- float [Accuracy](#)
This is the neighborhood around the target coordinates, every point in which is treated as the target position.

5.59.1 Detailed Description

Synchronization settings which use user units.

This structure contains all synchronization settings, modes, periods and flags. It specifies the behavior of the output synchronization. All boards are supplied with the standard set of these settings.

See also

[get_sync_out_settings_calb](#)
[set_sync_out_settings_calb](#)
[get_sync_out_settings](#), [set_sync_out_settings](#)

5.59.2 Field Documentation

5.59.2.1 Accuracy

float Accuracy

This is the neighborhood around the target coordinates, every point in which is treated as the target position.

Getting in these points cause the stop impulse.

5.59.2.2 SyncOutFlags

unsigned int SyncOutFlags

Flags of synchronization output.

5.59.2.3 SyncOutPeriod

unsigned int SyncOutPeriod

This value specifies the number of encoder pulses or steps between two output synchronization pulses when SYNCOUT_ONPERIOD is set.

5.59.2.4 SyncOutPulseSteps

unsigned int SyncOutPulseSteps

This value specifies the duration of output pulse.

It is measured microseconds when SYNCOUT_IN_STEPS flag is cleared or in encoder pulses or motor steps when SYNCOUT_IN_STEPS is set.

5.60 sync_out_settings_t Struct Reference

Synchronization settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int [SyncOutFlags](#)
Flags of synchronization output.
- unsigned int [SyncOutPulseSteps](#)
This value specifies the duration of output pulse.
- unsigned int [SyncOutPeriod](#)
This value specifies the number of encoder pulses or steps between two output synchronization pulses when SYNCOUT_ONPERIOD is set.
- unsigned int [Accuracy](#)
This is the neighborhood around the target coordinates, every point in which is treated as the target position.
- unsigned int [uAccuracy](#)
This is the neighborhood around the target coordinates in microsteps (used with a stepper motor only).

5.60.1 Detailed Description

Synchronization settings.

This structure contains all synchronization settings, modes, periods and flags. It specifies the behavior of the output synchronization. All boards are supplied with the standard set of these settings.

See also

[get_sync_out_settings](#)

[set_sync_out_settings](#)

[get_sync_out_settings](#), [set_sync_out_settings](#)

5.60.2 Field Documentation

5.60.2.1 Accuracy

`unsigned int Accuracy`

This is the neighborhood around the target coordinates, every point in which is treated as the target position.

Getting in these points cause the stop impulse.

5.60.2.2 SyncOutFlags

`unsigned int SyncOutFlags`

[Flags of synchronization output.](#)

5.60.2.3 SyncOutPeriod

`unsigned int SyncOutPeriod`

This value specifies the number of encoder pulses or steps between two output synchronization pulses when SYNCOUT_ONPERIOD is set.

5.60.2.4 SyncOutPulseSteps

`unsigned int SyncOutPulseSteps`

This value specifies the duration of output pulse.

It is measured microseconds when SYNCOUT_IN_STEPS flag is cleared or in encoder pulses or motor steps when SYNCOUT_IN_STEPS is set.

5.60.2.5 `uAccuracy`

`unsigned int uAccuracy`

This is the neighborhood around the target coordinates in microsteps (used with a stepper motor only).

The microstep size and the range of valid values for this field depend on the selected step division mode (see the `MicrostepMode` field in `engine_settings`).

5.61 `uart_settings_t` Struct Reference

UART settings.

```
#include <ximc.h>
```

Data Fields

- unsigned int **Speed**
UART baudrate (in bauds)
- unsigned int **UARTSetupFlags**
UART parity flags.

5.61.1 Detailed Description

UART settings.

This structure contains UART settings.

See also

[get_uart_settings](#)

[set_uart_settings](#)

[get_uart_settings](#), [set_uart_settings](#)

5.61.2 Field Documentation

5.61.2.1 **UARTSetupFlags**

`unsigned int UARTSetupFlags`

[UART parity flags.](#)

Chapter 6

File Documentation

6.1 ximc.h File Reference

Data Structures

- struct [calibration_t](#)
Calibration structure
- struct [device_network_information_t](#)
Device network information structure.
- struct [feedback_settings_t](#)
Feedback settings.
- struct [home_settings_t](#)
Position calibration settings.
- struct [home_settings_calb_t](#)
Position calibration settings which use user units.
- struct [move_settings_t](#)
Move settings.
- struct [move_settings_calb_t](#)
User units move settings.
- struct [engine_settings_t](#)
Movement limitations and settings related to the motor.
- struct [engine_settings_calb_t](#)
Movement limitations and settings, related to the motor.
- struct [entype_settings_t](#)
Engine type and driver type settings.
- struct [power_settings_t](#)
Step motor power settings.
- struct [secure_settings_t](#)
This structure contains protection parameters: critical electrical values, flags for protection algorithms.
- struct [edges_settings_t](#)
Edges settings.
- struct [edges_settings_calb_t](#)
User unit edges settings.
- struct [pid_settings_t](#)
PID settings.

- struct [sync_in_settings_t](#)
Synchronization settings.
- struct [sync_in_settings_calb_t](#)
User unit synchronization settings.
- struct [sync_out_settings_t](#)
Synchronization settings.
- struct [sync_out_settings_calb_t](#)
Synchronization settings which use user units.
- struct [extio_settings_t](#)
EXTIO settings.
- struct [brake_settings_t](#)
Brake settings.
- struct [control_settings_t](#)
Control settings.
- struct [control_settings_calb_t](#)
User unit control settings.
- struct [joystick_settings_t](#)
Joystick settings.
- struct [ctp_settings_t](#)
Control position settings (used with stepper motor only)
- struct [uart_settings_t](#)
UART settings.
- struct [network_settings_t](#)
Network settings.
- struct [password_settings_t](#)
The web-page user password.
- struct [calibration_settings_t](#)
Calibration settings.
- struct [controller_name_t](#)
Controller name and settings flags
- struct [nonvolatile_memory_t](#)
Structure contains user data to save into the FRAM.
- struct [emf_settings_t](#)
EMF settings.
- struct [engine_advanced_setup_t](#)
EAS settings.
- struct [extended_settings_t](#)
EST settings This data is stored in the controller's flash memory.
- struct [get_position_t](#)
Position information.
- struct [get_position_calb_t](#)
Position information.
- struct [set_position_t](#)
Position information.
- struct [set_position_calb_t](#)
User unit position information.
- struct [status_t](#)
Device state.
- struct [status_calb_t](#)

User unit device's state.

- struct [measurements_t](#)

The structure contains a sequence of measured axis motion parameters - velocities and position errors.

- struct [chart_data_t](#)

Additional device state.

- struct [device_information_t](#)

Controller information structure.

- struct [serial_number_t](#)

The structure contains a new serial number, hardware version, and valid key.

- struct [analog_data_t](#)

Analog data.

- struct [debug_read_t](#)

Debug data.

- struct [debug_write_t](#)

Debug data.

- struct [stage_name_t](#)

Stage username.

- struct [stage_information_t](#)

Deprecated.

- struct [stage_settings_t](#)

Deprecated.

- struct [motor_information_t](#)

Deprecated.

- struct [motor_settings_t](#)

Deprecated.

- struct [encoder_information_t](#)

Deprecated.

- struct [encoder_settings_t](#)

Deprecated.

- struct [hallsensor_information_t](#)

Deprecated.

- struct [hallsensor_settings_t](#)

Deprecated.

- struct [gear_information_t](#)

Deprecated.

- struct [gear_settings_t](#)

Deprecated.

- struct [accessories_settings_t](#)

Deprecated.

- struct [init_random_t](#)

Random key.

- struct [globally_unique_identifier_t](#)

Globally unique identifier.

Macros

- #define [XIMC_API](#)
- #define [XIMC_CALLCONV](#)
- #define [XIMC_RETTYPE](#) void*
- #define **device_undefined** -1
Handle specified undefined device

Result statuses

- #define **result_ok** 0
success
- #define **result_error** -1
generic error
- #define **result_not_implemented** -2
function is not implemented
- #define **result_value_error** -3
value error
- #define **result_nodvice** -4
device is lost

Enumerate devices flags

This is a bit mask for bitwise operations.

- #define [ENUMERATE_PROBE](#) 0x01
Check if a device with an OS name is a XIMC device.
- #define **ENUMERATE_ALL_COM** 0x02
Check all COM devices
- #define **ENUMERATE_NETWORK** 0x04
Check network devices

Flags of move state

This is a bit mask for bitwise operations.

Specify move states.

See also

[get_status](#)
[status_t::MoveSts](#), [get_status_impl](#)

- #define [MOVE_STATE_MOVING](#) 0x01
This flag indicates that the controller is trying to move the motor.
- #define [MOVE_STATE_TARGET_SPEED](#) 0x02
Target speed is reached, if flag set.
- #define [MOVE_STATE_ANTIPLAY](#) 0x04
Motor is playing compensation, if flag set.

Flags of internal controller settings

This is a bit mask for bitwise operations.

See also

[set_controller_name](#)
[get_controller_name](#)
[controller_name_t::CtrlFlags](#), [get_controller_name](#), [set_controller_name](#)

- #define [EEPROM_PRECEDENCE](#) 0x01
If the flag is set, settings from external EEPROM override controller settings.

Flags of power state of stepper motor

This is a bit mask for bitwise operations.

Specify power states.

See also

[get_status](#)
[status_t::PWRSts](#), [get_status_impl](#)

- #define [PWR_STATE_UNKNOWN](#) 0x00
Unknown state, should never happen.
- #define [PWR_STATE_OFF](#) 0x01
Motor windings are disconnected from the driver.
- #define [PWR_STATE_NORM](#) 0x03
Motor windings are powered by nominal current.
- #define [PWR_STATE_REDUCT](#) 0x04
Motor windings are powered by reduced current to lower power consumption.
- #define [PWR_STATE_MAX](#) 0x05
Motor windings are powered by the maximum current driver can provide at this voltage.

Status flags

This is a bit mask for bitwise operations.

Controller flags returned by device query. Contains boolean part of controller state. May be combined with bitwise OR.

See also

[get_status](#)
[status_t::Flags](#), [get_status_impl](#)

- #define [STATE_CONTR](#) 0x000003F
Flags of controller states.
- #define [STATE_ERRC](#) 0x0000001
Command error encountered.
- #define [STATE_ERRD](#) 0x0000002
Data integrity error encountered.
- #define [STATE_ERRV](#) 0x0000004
Value error encountered.
- #define [STATE_EEPROM_CONNECTED](#) 0x0000010
EEPROM with settings is connected.
- #define [STATE_IS_HOMED](#) 0x0000020
Calibration performed.
- #define [STATE_SECUR](#) 0x1B3FFC0
Security flags.
- #define [STATE_ALARM](#) 0x0000040
The controller is in an alarm state, indicating that something dangerous has happened.
- #define [STATE_CTP_ERROR](#) 0x0000080
Control position error (is only used with stepper motor).

- #define [STATE_POWER_OVERHEAT](#) 0x0000100
Power driver overheat.
- #define [STATE_CONTROLLER_OVERHEAT](#) 0x0000200
Controller overheat.
- #define [STATE_OVERLOAD_POWER_VOLTAGE](#) 0x0000400
Power voltage exceeds safe limit.
- #define [STATE_OVERLOAD_POWER_CURRENT](#) 0x0000800
Power current exceeds safe limit.
- #define [STATE_OVERLOAD_USB_VOLTAGE](#) 0x0001000
Deprecated.
- #define [STATE_LOW_USB_VOLTAGE](#) 0x0002000
Deprecated.
- #define [STATE_OVERLOAD_USB_CURRENT](#) 0x0004000
Deprecated.
- #define [STATE_BORDERS_SWAP_MISSET](#) 0x0008000
Engine stuck at the wrong edge.
- #define **STATE_LOW_POWER_VOLTAGE** 0x0010000
Power voltage is lower than Low Voltage Protection limit
- #define **STATE_H_BRIDGE_FAULT** 0x0020000
Signal from the driver that fault happened
- #define [STATE_WINDING_RES_MISMATCH](#) 0x0100000
The difference between winding resistances is too large.
- #define **STATE_ENCODER_FAULT** 0x0200000
Signal from the encoder that fault happened
- #define [STATE_ENGINE_RESPONSE_ERROR](#) 0x0800000
Error response of the engine control action.
- #define [STATE_EXTIO_ALARM](#) 0x1000000
The error is caused by the external EXTIO input signal.

Status flags of the GPIO outputs

This is a bit mask for bitwise operations.

GPIO state flags returned by device query. Contains boolean part of controller state. May be combined with bitwise OR.

See also

[get_status](#)

[status.t::GPIOFlags](#), [get_status_impl](#)

- #define [STATE_DIG_SIGNAL](#) 0xFFFF
Flags of digital signals.
- #define [STATE_RIGHT_EDGE](#) 0x0001
Engine stuck at the right edge.
- #define [STATE_LEFT_EDGE](#) 0x0002
Engine stuck at the left edge.
- #define [STATE_BUTTON_RIGHT](#) 0x0004
Button "right" state (1 if pressed).
- #define [STATE_BUTTON_LEFT](#) 0x0008
Button "left" state (1 if pressed).
- #define [STATE_GPIO_PINOUT](#) 0x0010
External GPIO works as out if the flag is set; otherwise, it works as in.
- #define [STATE_GPIO_LEVEL](#) 0x0020
State of external GPIO pin.
- #define [STATE_BRAKE](#) 0x0200
State of Brake pin.
- #define [STATE_REV_SENSOR](#) 0x0400
State of Revolution sensor pin.

- #define [STATE_SYNC_INPUT](#) 0x0800
State of Sync input pin.
- #define [STATE_SYNC_OUTPUT](#) 0x1000
State of Sync output pin.
- #define [STATE_ENC_A](#) 0x2000
State of encoder A pin.
- #define [STATE_ENC_B](#) 0x4000
State of encoder B pin.

Encoder state

This is a bit mask for bitwise operations.

Encoder state returned by device query.

Note

The ENCD flag is only active in the None and EMF modes. In other modes, such as Encoder and Encoder Mediated, it is not used, as these modes cannot operate without an encoder. After the controller is powered on, the flag will be in the unknown state — a movement is required to determine the position. As the movement occurs, the flag's value may change.

See also

[get_status](#)

[status_t::EncSts](#), [get_status_impl](#)

- #define [ENC_STATE_ABSENT](#) 0x00
Encoder is absent.
- #define [ENC_STATE_UNKNOWN](#) 0x01
Encoder state is unknown.
- #define [ENC_STATE_MALFUNC](#) 0x02
Encoder is connected and malfunctioning.
- #define [ENC_STATE_REVERS](#) 0x03
Encoder is connected and operational but counts in other direction.
- #define [ENC_STATE_OK](#) 0x04
Encoder is connected and working properly.

Winding state

This is a bit mask for bitwise operations.

Motor winding state returned by device query.

See also

[get_status](#)

[status_t::WindSts](#), [get_status_impl](#)

- #define [WIND_A_STATE_ABSENT](#) 0x00
Winding A is disconnected.
- #define [WIND_A_STATE_UNKNOWN](#) 0x01
Winding A state is unknown.
- #define [WIND_A_STATE_MALFUNC](#) 0x02
Winding A is short-circuited.
- #define [WIND_A_STATE_OK](#) 0x03
Winding A is connected and working properly.
- #define [WIND_B_STATE_ABSENT](#) 0x00
Winding B is disconnected.
- #define [WIND_B_STATE_UNKNOWN](#) 0x10

- *Winding B state is unknown.*
• #define [WIND_B_STATE_MALFUNC](#) 0x20
- *Winding B is short-circuited.*
• #define [WIND_B_STATE_OK](#) 0x30
- *Winding B is connected and working properly.*

Move command state

This is a bit mask for bitwise operations.

Move command (command_move, command_movr, command_left, command_right, command_stop, command_home, command_loft, command_sstp) and its state (run, finished, error).

See also

[get_status](#)
[status_t::MvCmdSts](#), [get_status_impl](#)

- #define [MVCMD_NAME_BITS](#) 0x3F
Move command bit mask.
- #define [MVCMD_UKNWN](#) 0x00
Unknown command.
- #define [MVCMD_MOVE](#) 0x01
Command move.
- #define [MVCMD_MOVR](#) 0x02
Command movr.
- #define [MVCMD_LEFT](#) 0x03
Command left.
- #define [MVCMD_RIGHT](#) 0x04
Command right.
- #define [MVCMD_STOP](#) 0x05
Command stop.
- #define [MVCMD_HOME](#) 0x06
Command home.
- #define [MVCMD_LOFT](#) 0x07
Command loft.
- #define [MVCMD_SSTP](#) 0x08
Command soft stop.
- #define [MVCMD_ERROR](#) 0x40
Finish state (1 - move command has finished with an error, 0 - move command has finished correctly).
- #define [MVCMD_RUNNING](#) 0x80
Move command state (0 - move command has finished, 1 - move command is being executed).

Flags of the motion parameters

This is a bit mask for bitwise operations.

Specify the motor shaft movement algorithm and list of limitations. Flags returned by the query of [get_move_settings](#).

See also

[set_move_settings](#)
[get_move_settings](#)
[move_settings_t::MoveFlags](#), [get_move_settings](#), [set_move_settings](#)

- #define [RPM_DIV_1000](#) 0x01
This flag indicates that the operating speed specified in the command is set in milliRPM.

Flags of engine settings

This is a bit mask for bitwise operations.

Specify the motor shaft movement algorithm and list of limitations. Flags returned by query of engine settings. May be combined with bitwise OR.

See also

[set_engine_settings](#)
[get_engine_settings](#)
[engine_settings_t::EngineFlags](#), [get_engine_settings](#), [set_engine_settings](#)

- #define [ENGINE_REVERSE](#) 0x01
Reverse flag.
- #define [ENGINE_CURRENT_AS_RMS](#) 0x02
Engine current meaning flag.
- #define [ENGINE_MAX_SPEED](#) 0x04
Max speed flag.
- #define [ENGINE_ANTIPLAY](#) 0x08
Play compensation flag.
- #define [ENGINE_ACCEL_ON](#) 0x10
Acceleration enable flag.
- #define [ENGINE_LIMIT_VOLT](#) 0x20
Maximum motor voltage limit enable flag (is only used with DC motor).
- #define [ENGINE_LIMIT_CURR](#) 0x40
Maximum motor current limit enable flag (is only used with DC motor).
- #define [ENGINE_LIMIT_RPM](#) 0x80
Maximum motor speed limit enable flag.

Flags of microstep mode

This is a bit mask for bitwise operations.

Specify settings for microstep mode. Used with step motors. Flags returned by query of engine settings. May be combined with bitwise OR

See also

[engine_settings_t::flags](#)
[set_engine_settings](#)
[get_engine_settings](#)
[engine_settings_t::MicrostepMode](#), [get_engine_settings](#), [set_engine_settings](#)

- #define [MICROSTEP_MODE_FULL](#) 0x01
Full step mode.
- #define [MICROSTEP_MODE_FRAC_2](#) 0x02
1/2-step mode.
- #define [MICROSTEP_MODE_FRAC_4](#) 0x03
1/4-step mode.
- #define [MICROSTEP_MODE_FRAC_8](#) 0x04
1/8-step mode.
- #define [MICROSTEP_MODE_FRAC_16](#) 0x05
1/16-step mode.
- #define [MICROSTEP_MODE_FRAC_32](#) 0x06
1/32-step mode.
- #define [MICROSTEP_MODE_FRAC_64](#) 0x07
1/64-step mode.
- #define [MICROSTEP_MODE_FRAC_128](#) 0x08
1/128-step mode.
- #define [MICROSTEP_MODE_FRAC_256](#) 0x09
1/256-step mode.

Flags of engine type

This is a bit mask for bitwise operations.

Specify motor type. Flags returned by query of engine settings.

See also

[engine_settings_t::flags](#)
[set_entype_settings](#)
[get_entype_settings](#)
[entype_settings_t::EngineType](#), [get_entype_settings](#), [set_entype_settings](#)

- #define [ENGINE_TYPE_NONE](#) 0x00
A value that shouldn't be used.
- #define [ENGINE_TYPE_DC](#) 0x01
DC motor.
- #define [ENGINE_TYPE_2DC](#) 0x02
2 DC motors.
- #define [ENGINE_TYPE_STEP](#) 0x03
Step motor.
- #define [ENGINE_TYPE_TEST](#) 0x04
Duty cycle are fixed.
- #define [ENGINE_TYPE_BRUSHLESS](#) 0x05
Brushless motor.

Flags of driver type

This is a bit mask for bitwise operations.

Specify driver type. Flags returned by query of engine settings.

See also

[engine_settings_t::flags](#)
[set_entype_settings](#)
[get_entype_settings](#)
[entype_settings_t::DriverType](#), [get_entype_settings](#), [set_entype_settings](#)

- #define [DRIVER_TYPE_INTEGRATE](#) 0x02
Driver with integrated IC.
- #define [DRIVER_TYPE_EXTERNAL](#) 0x03
External driver.

Flags of power settings of stepper motor

This is a bit mask for bitwise operations.

Flags returned by query of engine settings. Specify power settings. Flags returned by query of power settings.

See also

[get_power_settings](#)
[set_power_settings](#)
[power_settings_t::PowerFlags](#), [get_power_settings](#), [set_power_settings](#)

- #define [POWER_REDUCT_ENABLED](#) 0x01
Current reduction is enabled after CurrReductDelay if this flag is set.
- #define [POWER_OFF_ENABLED](#) 0x02
Power off is enabled after PowerOffDelay if this flag is set.
- #define [POWER_SMOOTH_CURRENT](#) 0x04
Current ramp-up/down are performed smoothly during current_set_time if this flag is set.

Flags of secure settings

This is a bit mask for bitwise operations.

Flags returned by query of engine settings. Specify secure settings. Flags returned by query of secure settings.

See also

[get_secure_settings](#)
[set_secure_settings](#)
[secure_settings_t::Flags](#), [get_secure_settings](#), [set_secure_settings](#)

- #define **ALARM_ON_DRIVER_OVERHEATING** 0x01
If this flag is set, enter the alarm state on the driver overheat signal.
- #define **LOW_UPWR_PROTECTION** 0x02
If this flag is set, turn off the motor when the voltage is lower than LowUpwrOff.
- #define **H_BRIDGE_ALERT** 0x04
If this flag is set, then turn off the power unit with a signal problem in one of the transistor bridge.
- #define **ALARM_ON_BORDERS_SWAP_MISSET** 0x08
If this flag is set, enter Alarm state on borders swap misset
- #define **ALARM_FLAGS_STICKING** 0x10
If this flag is set, only a STOP command can turn all alarms to 0
- #define **BRAKING_OVERTVOLTAGE_PROTECTION** 0x20
If this flag is set, the firmware will close ground switches of H-bridge to disconnect motor from power circuit on overvoltage.
- #define **ALARM_WINDING_MISMATCH** 0x40
If this flag is set, enter Alarm state when windings mismatch
- #define **ALARM_ENGINE_RESPONSE** 0x80
If this flag is set, enter the Alarm state on response of the engine control action

Position setting flags

This is a bit mask for bitwise operations.

Flags used in setting position.

See also

[get_position](#)
[set_position](#)
[set_position_t::PosFlags](#), [set_position](#)

- #define **SETPOS_IGNORE_POSITION** 0x01
Will not reload position in steps/microsteps if this flag is set.
- #define **SETPOS_IGNORE_ENCODER** 0x02
Will not reload encoder state if this flag is set.

Feedback type.

This is a bit mask for bitwise operations.

See also

[set_feedback_settings](#)
[get_feedback_settings](#)
[feedback_settings_t::FeedbackType](#), [get_feedback_settings](#), [set_feedback_settings](#)

- #define **FEEDBACK_ENCODER** 0x01
Feedback by encoder.
- #define **FEEDBACK_EMF** 0x04
Feedback by EMF.
- #define **FEEDBACK_NONE** 0x05
Feedback is absent.
- #define **FEEDBACK_ENCODER_MEDIATED** 0x06
Feedback by encoder mediated by mechanical transmission (for example leadscrew).

Describes feedback flags.

This is a bit mask for bitwise operations.

See also

[set_feedback_settings](#)
[get_feedback_settings](#)
[feedback_settings_t::FeedbackFlags](#), [get_feedback_settings](#), [set_feedback_settings](#)

- #define [FEEDBACK_ENC_REVERSE](#) 0x01
Reverse count of encoder.
- #define [FEEDBACK_ENC_ADAPTIVE_HOLDING](#) 0x02
Enables the adaptive holding algorithm.
- #define [FEEDBACK_ENC_FILTER_NONE](#) 0x00
Disable the internal filter of the encoder signal.
- #define [FEEDBACK_ENC_FILTER_WEAK](#) 0x10
Weak noise filtering: the maximum encoder signal frequency is 3 MHz.
- #define [FEEDBACK_ENC_FILTER_MEDIUM](#) 0x20
Medium noise filtering: the maximum encoder signal frequency is 1 MHz.
- #define [FEEDBACK_ENC_FILTER_STRONG](#) 0x30
Strong noise filtering: the maximum encoder signal frequency is 300 kHz.
- #define [FEEDBACK_ENC_FILTER_BITS](#) 0x30
Bits responsible for setting the internal filter of the encoder signal.
- #define [FEEDBACK_ENC_TYPE_AUTO](#) 0x00
Auto detect encoder type.
- #define [FEEDBACK_ENC_TYPE_SINGLE_ENDED](#) 0x40
Single-ended encoder.
- #define [FEEDBACK_ENC_TYPE_DIFFERENTIAL](#) 0x80
Differential encoder.
- #define [FEEDBACK_ENC_TYPE_BITS](#) 0xC0
Bits of the encoder type.

Flags for synchronization input setup

This is a bit mask for bitwise operations.

See also

[sync_in_settings_t::SyncInFlags](#), [get_sync_in_settings](#), [set_sync_in_settings](#)

- #define [SYNCIN_ENABLED](#) 0x01
Synchronization in mode is enabled if this flag is set.
- #define [SYNCIN_INVERT](#) 0x02
Trigger on falling edge if flag is set, on rising edge otherwise.
- #define [SYNCIN_GOTOPOSITION](#) 0x04
The engine is going to the position specified in Position and uPosition if this flag is set.

Flags of synchronization output

This is a bit mask for bitwise operations.

See also

[sync_out_settings_t::SyncOutFlags](#), [get_sync_out_settings](#), [set_sync_out_settings](#)

- #define [SYNCOUT_ENABLED](#) 0x01
The synchronization out pin follows the synchronization logic if the flag is set.
- #define [SYNCOUT_STATE](#) 0x02
When the output state is fixed by the negative SYNCOUT_ENABLED flag, the pin state is in accordance with this flag state.
- #define [SYNCOUT_INVERT](#) 0x04
The low level is active if the flag is set.
- #define [SYNCOUT_IN_STEPS](#) 0x08

- *Use motor steps or encoder pulses instead of milliseconds for output pulse generation if the flag is set.*
• `#define SYNCOUT_ONSTART 0x10`
Generate a synchronization pulse when movement starts.
- `#define SYNCOUT_ONSTOP 0x20`
Generate a synchronization pulse when movement stops.
- `#define SYNCOUT_ONPERIOD 0x40`
Generate a synchronization pulse every SyncOutPeriod encoder pulses.

External IO setup flags

This is a bit mask for bitwise operations.

See also

[get_extio_settings](#)
[set_extio_settings](#)
[extio_settings_t::EXTIOSetupFlags](#), [get_extio_settings](#), [set_extio_settings](#)

- `#define EXTIO_SETUP_OUTPUT 0x01`
EXTIO works as output if the flag is set, works as input otherwise.
- `#define EXTIO_SETUP_INVERT 0x02`
Interpret EXTIO state inverted if the flag is set.

External IO mode flags

This is a bit mask for bitwise operations.

See also

[extio_settings_t::extio_mode_flags](#)
[get_extio_settings](#)
[set_extio_settings](#)
[extio_settings_t::EXTIOModeFlags](#), [get_extio_settings](#), [set_extio_settings](#)

- `#define EXTIO_SETUP_MODE_IN_BITS 0x0F`
Bits of the behavior selector when the signal on input goes to the active state.
- `#define EXTIO_SETUP_MODE_IN_NOP 0x00`
Do nothing.
- `#define EXTIO_SETUP_MODE_IN_STOP 0x01`
Issue STOP command, ceasing the engine movement.
- `#define EXTIO_SETUP_MODE_IN_PWOF 0x02`
Issue PWOF command, powering off all engine windings.
- `#define EXTIO_SETUP_MODE_IN_MOVR 0x03`
Issue MOVR command with last used settings.
- `#define EXTIO_SETUP_MODE_IN_HOME 0x04`
Issue HOME command.
- `#define EXTIO_SETUP_MODE_IN_ALARM 0x05`
Set Alarm when the signal goes to the active state.
- `#define EXTIO_SETUP_MODE_OUT_BITS 0xF0`
Bits of the output behavior selection.
- `#define EXTIO_SETUP_MODE_OUT_OFF 0x00`
EXTIO pin always set in inactive state.
- `#define EXTIO_SETUP_MODE_OUT_ON 0x10`
EXTIO pin always set in active state.
- `#define EXTIO_SETUP_MODE_OUT_MOVING 0x20`
EXTIO pin stays active during moving state.
- `#define EXTIO_SETUP_MODE_OUT_ALARM 0x30`
EXTIO pin stays active during the alarm state.

- #define [EXTIO_SETUP_MODE_OUT_MOTOR_ON](#) 0x40
EXTIO pin stays active when windings are powered.

Border flags

This is a bit mask for bitwise operations.

Specify types of borders and motor behavior on borders. May be combined with bitwise OR.

See also

[get_edges_settings](#)
[set_edges_settings](#)
[edges_settings_t::BorderFlags](#), [get_edges_settings](#), [set_edges_settings](#)

- #define [BORDER_IS_ENCODER](#) 0x01
Borders are fixed by predetermined encoder values, if set; borders are placed on limit switches, if not set.
- #define [BORDER_STOP_LEFT](#) 0x02
The motor should stop on the left border.
- #define [BORDER_STOP_RIGHT](#) 0x04
Motor should stop on right border.
- #define [BORDERS_SWAP_MISSET_DETECTION](#) 0x08
Motor should stop on both borders.

Limit switches flags

This is a bit mask for bitwise operations.

Specify electrical behavior of limit switches like order and pulled positions. May be combined with bitwise OR.

See also

[get_edges_settings](#)
[set_edges_settings](#)
[edges_settings_t::EnderFlags](#), [get_edges_settings](#), [set_edges_settings](#)

- #define [ENDER_SWAP](#) 0x01
First limit switch on the right side, if set; otherwise on the left side.
- #define [ENDER_SW1_ACTIVE_LOW](#) 0x02
1 - Limit switch connected to pin SW1 is triggered by a low level on pin.
- #define [ENDER_SW2_ACTIVE_LOW](#) 0x04
1 - Limit switch connected to pin SW2 is triggered by a low level on pin.

Brake settings flags

This is a bit mask for bitwise operations.

Specify behavior of brake. May be combined with bitwise OR.

See also

[get_brake_settings](#)
[set_brake_settings](#)
[brake_settings_t::BrakeFlags](#), [get_brake_settings](#), [set_brake_settings](#)

- #define [BRAKE_ENABLED](#) 0x01
Brake control is enabled if this flag is set.
- #define [BRAKE_ENG_PWROFF](#) 0x02
Brake turns the stepper motor power off if this flag is set.

Control flags

This is a bit mask for bitwise operations.

Specify motor control settings by joystick or buttons. May be combined with bitwise OR.

See also

[get_control_settings](#)
[set_control_settings](#)
[control_settings.t::Flags](#), [get_control_settings](#), [set_control_settings](#)

- #define [CONTROL_MODE_BITS](#) 0x03
Bits to control the engine by joystick or buttons.
- #define [CONTROL_MODE_OFF](#) 0x00
Control is disabled.
- #define [CONTROL_MODE_JOY](#) 0x01
Control by joystick.
- #define [CONTROL_MODE_LR](#) 0x02
Control by left/right buttons.
- #define [CONTROL_BTN_LEFT_PUSHED_OPEN](#) 0x04
Pushed left button corresponds to the open contact if this flag is set.
- #define [CONTROL_BTN_RIGHT_PUSHED_OPEN](#) 0x08
Pushed right button corresponds to open contact if this flag is set.

Joystick flags

This is a bit mask for bitwise operations.

Control joystick states.

See also

[set_joystick_settings](#)
[get_joystick_settings](#)
[joystick_settings.t::JoyFlags](#), [get_joystick_settings](#), [set_joystick_settings](#)

- #define [JOY_REVERSE](#) 0x01
Joystick action is reversed.

Position control flags

This is a bit mask for bitwise operations.

Specify control position settings. May be combined with bitwise OR.

See also

[get_ctp_settings](#)
[set_ctp_settings](#)
[ctp_settings.t::CTPFlags](#), [get_ctp_settings](#), [set_ctp_settings](#)

- #define [CTP_ENABLED](#) 0x01
The position control is enabled if the flag is set.
- #define [CTP_BASE](#) 0x02
The position control is based on the revolution sensor if this flag is set; otherwise, it is based on the encoder.
- #define [CTP_ALARM_ON_ERROR](#) 0x04
Set ALARM on mismatch if the flag is set.
- #define [REV_SENS_INV](#) 0x08
Typically, the sensor is active when it is at 0, and inversion makes active at 1.
- #define [CTP_ERROR_CORRECTION](#) 0x10
Correct errors that appear when slippage occurs if the flag is set.

Home settings flags

This is a bit mask for bitwise operations.

Specify home command behavior. May be combined with bitwise OR.

See also

[get_home_settings](#)
[set_home_settings](#)
[command_home](#)
[home_settings_t::HomeFlags](#), [get_home_settings](#), [set_home_settings](#)

- #define [HOME_DIR_FIRST](#) 0x001
The flag defines the direction of the 1st motion after execution of the home command.
- #define [HOME_DIR_SECOND](#) 0x002
The flag defines the direction of the 2nd motion.
- #define [HOME_MV_SEC_EN](#) 0x004
Use the second phase of calibration to the home position, if set; otherwise the second phase is skipped.
- #define [HOME_HALF_MV](#) 0x008
If the flag is set, the stop signals are ignored during the first half-turn of the second movement.
- #define [HOME_STOP_FIRST_BITS](#) 0x030
Bits of the first stop selector.
- #define [HOME_STOP_FIRST_REV](#) 0x010
First motion stops by revolution sensor.
- #define [HOME_STOP_FIRST_SYN](#) 0x020
First motion stops by synchronization input.
- #define [HOME_STOP_FIRST_LIM](#) 0x030
First motion stops by limit switch.
- #define [HOME_STOP_SECOND_BITS](#) 0x0C0
Bits of the second stop selector.
- #define [HOME_STOP_SECOND_REV](#) 0x040
Second motion stops by revolution sensor.
- #define [HOME_STOP_SECOND_SYN](#) 0x080
Second motion stops by synchronization input.
- #define [HOME_STOP_SECOND_LIM](#) 0x0C0
Second motion stops by limit switch.
- #define [HOME_USE_FAST](#) 0x100
Use the fast algorithm of calibration to the home position, if set; otherwise the traditional algorithm.

UART parity flags

This is a bit mask for bitwise operations.

See also

[uart_settings_t::UARTSetupFlags](#), [get_uart_settings](#), [set_uart_settings](#)

- #define [UART_PARITY_BITS](#) 0x03
Bits of the parity.
- #define [UART_PARITY_BIT_EVEN](#) 0x00
Parity bit 1, if even
- #define [UART_PARITY_BIT_ODD](#) 0x01
Parity bit 1, if odd
- #define [UART_PARITY_BIT_SPACE](#) 0x02
Parity bit always 0
- #define [UART_PARITY_BIT_MARK](#) 0x03
Parity bit always 1
- #define [UART_PARITY_BIT_USE](#) 0x04
None parity
- #define [UART_STOP_BIT](#) 0x08
If set - one stop bit, else two stop bit

Motor Type flags

This is a bit mask for bitwise operations.

See also

[motor_settings_t::MotorType](#), [get_motor_settings](#), [set_motor_settings](#)

- #define **MOTOR_TYPE_UNKNOWN** 0x00
Unknown type of engine
- #define **MOTOR_TYPE_STEP** 0x01
Step engine
- #define **MOTOR_TYPE_DC** 0x02
DC engine
- #define **MOTOR_TYPE_BLDC** 0x03
BLDC engine

Encoder settings flags

This is a bit mask for bitwise operations.

See also

[encoder_settings_t::EncoderSettings](#), [get_encoder_settings](#), [set_encoder_settings](#)

- #define **ENCSET_DIFFERENTIAL_OUTPUT** 0x001
If the flag is set, the encoder has differential output, otherwise single-ended output
- #define **ENCSET_PUSH_PULL_OUTPUT** 0x004
If the flag is set the encoder has push-pull output, otherwise open drain output
- #define **ENCSET_INDEXCHANNEL_PRESENT** 0x010
If the flag is set, the encoder has an extra indexed channel
- #define **ENCSET_REVOLUTIONSENSOR_PRESENT** 0x040
If the flag is set, the encoder has the revolution sensor
- #define **ENCSET_REVOLUTIONSENSOR_ACTIVE_HIGH** 0x100
If the flag is set, the revolution sensor's active state is high logic state; otherwise, the active state is low logic state

Magnetic brake settings flags

This is a bit mask for bitwise operations.

See also

[accessories_settings_t::MBSettings](#), [get_accessories_settings](#), [set_accessories_settings](#)

- #define **MB_AVAILABLE** 0x01
If the flag is set, the magnetic brake is available
- #define **MB_POWERED_HOLD** 0x02
If this flag is set, the magnetic brake is on when powered

Temperature sensor settings flags

This is a bit mask for bitwise operations.

See also

[accessories_settings_t::LimitSwitchesSettings](#), [get_accessories_settings](#), [set_accessories_settings](#)

- #define **TS_TYPE_BITS** 0x07
Bits of the temperature sensor type
- #define **TS_TYPE_UNKNOWN** 0x00
Unknown type of sensor
- #define **TS_TYPE_THERMOCOUPLE** 0x01
Thermocouple
- #define **TS_TYPE_SEMICONDUCTOR** 0x02

- *The semiconductor temperature sensor*
- `#define TS_AVAILABLE 0x08`
If the flag is set, the temperature sensor is available
- `#define LS_ON_SW1_AVAILABLE 0x01`
If the flag is set, the limit switch connected to pin SW1 is available
- `#define LS_ON_SW2_AVAILABLE 0x02`
If the flag is set, the limit switch connected to pin SW2 is available
- `#define LS_SW1_ACTIVE_LOW 0x04`
If the flag is set, the limit switch connected to pin SW1 is triggered by a low level on the pin
- `#define LS_SW2_ACTIVE_LOW 0x08`
If the flag is set, the limit switch connected to pin SW2 is triggered by a low level on pin
- `#define LS_SHORTED 0x10`
If the flag is set, the limit switches are shorted

Flags of auto-detection of characteristics of windings of the engine.

This is a bit mask for bitwise operations.

See also

[set_emf_settings](#)
[get_emf_settings](#)
[emf_settings_t::BackEMFFlags](#), [get_emf_settings](#), [set_emf_settings](#)

- `#define BACK_EMF_INDUCTANCE_AUTO 0x01`
Flag of auto-detection of inductance of windings of the engine.
- `#define BACK_EMF_RESISTANCE_AUTO 0x02`
Flag of auto-detection of resistance of windings of the engine.
- `#define BACK_EMF_KM_AUTO 0x04`
Flag of auto-detection of electromechanical coefficient of the engine.

Typedefs

- `typedef unsigned long long ulong_t`
- `typedef long long long_t`
- `typedef int device_t`
Type describes device identifier
- `typedef int result_t`
Type specifies result of any operation
- `typedef uint32_t device_enumeration_t`
Type describes device enumeration structure

Functions

Controller settings setup

Read and write functions for almost all controller settings.

- `result_t XIMC_API set_feedback_settings (device_t id, const feedback_settings_t *feedback_settings)`
Feedback settings.
- `result_t XIMC_API get_feedback_settings (device_t id, feedback_settings_t *feedback_settings)`
Feedback settings.
- `result_t XIMC_API set_home_settings (device_t id, const home_settings_t *home_settings)`
Set home settings.

- [result_t XIMC_API set_home_settings_calb](#) ([device_t](#) id, const [home_settings_calb_t](#) *home_settings_calb, const [calibration_t](#) *calibration)
Set user unit home settings.
- [result_t XIMC_API get_home_settings](#) ([device_t](#) id, [home_settings_t](#) *home_settings)
Read home settings.
- [result_t XIMC_API get_home_settings_calb](#) ([device_t](#) id, [home_settings_calb_t](#) *home_settings_calb, const [calibration_t](#) *calibration)
Read user unit home settings.
- [result_t XIMC_API set_move_settings](#) ([device_t](#) id, const [move_settings_t](#) *move_settings)
Movement settings set command (speed, acceleration, threshold, etc.).
- [result_t XIMC_API set_move_settings_calb](#) ([device_t](#) id, const [move_settings_calb_t](#) *move_settings_calb, const [calibration_t](#) *calibration)
User unit movement settings set command (speed, acceleration, threshold, etc.).
- [result_t XIMC_API get_move_settings](#) ([device_t](#) id, [move_settings_t](#) *move_settings)
Movement settings read command (speed, acceleration, threshold, etc.).
- [result_t XIMC_API get_move_settings_calb](#) ([device_t](#) id, [move_settings_calb_t](#) *move_settings_calb, const [calibration_t](#) *calibration)
User unit movement settings read command (speed, acceleration, threshold, etc.).
- [result_t XIMC_API set_engine_settings](#) ([device_t](#) id, const [engine_settings_t](#) *engine_settings)
Set engine settings.
- [result_t XIMC_API set_engine_settings_calb](#) ([device_t](#) id, const [engine_settings_calb_t](#) *engine_settings_calb, const [calibration_t](#) *calibration)
Set user unit engine settings.
- [result_t XIMC_API get_engine_settings](#) ([device_t](#) id, [engine_settings_t](#) *engine_settings)
Read engine settings.
- [result_t XIMC_API get_engine_settings_calb](#) ([device_t](#) id, [engine_settings_calb_t](#) *engine_settings_calb, const [calibration_t](#) *calibration)
Read user unit engine settings.
- [result_t XIMC_API set_entype_settings](#) ([device_t](#) id, const [entype_settings_t](#) *entype_settings)
Set engine type and driver type.
- [result_t XIMC_API get_entype_settings](#) ([device_t](#) id, [entype_settings_t](#) *entype_settings)
Return engine type and driver type.
- [result_t XIMC_API set_power_settings](#) ([device_t](#) id, const [power_settings_t](#) *power_settings)
Set settings of step motor power control.
- [result_t XIMC_API get_power_settings](#) ([device_t](#) id, [power_settings_t](#) *power_settings)
Read settings of step motor power control.
- [result_t XIMC_API set_secure_settings](#) ([device_t](#) id, const [secure_settings_t](#) *secure_settings)
Set protection settings.
- [result_t XIMC_API get_secure_settings](#) ([device_t](#) id, [secure_settings_t](#) *secure_settings)
Read protection settings.
- [result_t XIMC_API set_edges_settings](#) ([device_t](#) id, const [edges_settings_t](#) *edges_settings)
Set border and limit switches settings.
- [result_t XIMC_API set_edges_settings_calb](#) ([device_t](#) id, const [edges_settings_calb_t](#) *edges_settings_calb, const [calibration_t](#) *calibration)
Set border and limit switches settings in user units.
- [result_t XIMC_API get_edges_settings](#) ([device_t](#) id, [edges_settings_t](#) *edges_settings)
Read border and limit switches settings.
- [result_t XIMC_API get_edges_settings_calb](#) ([device_t](#) id, [edges_settings_calb_t](#) *edges_settings_calb, const [calibration_t](#) *calibration)
Read border and limit switches settings in user units.
- [result_t XIMC_API set_pid_settings](#) ([device_t](#) id, const [pid_settings_t](#) *pid_settings)
Set PID settings.
- [result_t XIMC_API get_pid_settings](#) ([device_t](#) id, [pid_settings_t](#) *pid_settings)
Read PID settings.
- [result_t XIMC_API set_sync_in_settings](#) ([device_t](#) id, const [sync_in_settings_t](#) *sync_in_settings)
Set input synchronization settings.

- [result_t XIMC_API set_sync_in_settings_calb](#) ([device_t](#) id, const [sync_in_settings_calb_t](#) *sync_in_settings_calb, const [calibration_t](#) *calibration)
Set input user unit synchronization settings.
- [result_t XIMC_API get_sync_in_settings](#) ([device_t](#) id, [sync_in_settings_t](#) *sync_in_settings)
Read input synchronization settings.
- [result_t XIMC_API get_sync_in_settings_calb](#) ([device_t](#) id, [sync_in_settings_calb_t](#) *sync_in_settings_calb, const [calibration_t](#) *calibration)
Read input user unit synchronization settings.
- [result_t XIMC_API set_sync_out_settings](#) ([device_t](#) id, const [sync_out_settings_t](#) *sync_out_settings)
Set output synchronization settings.
- [result_t XIMC_API set_sync_out_settings_calb](#) ([device_t](#) id, const [sync_out_settings_calb_t](#) *sync_out_settings_calb, const [calibration_t](#) *calibration)
Set output user unit synchronization settings.
- [result_t XIMC_API get_sync_out_settings](#) ([device_t](#) id, [sync_out_settings_t](#) *sync_out_settings)
Read output synchronization settings.
- [result_t XIMC_API get_sync_out_settings_calb](#) ([device_t](#) id, [sync_out_settings_calb_t](#) *sync_out_settings_calb, const [calibration_t](#) *calibration)
Read output user unit synchronization settings.
- [result_t XIMC_API set_extio_settings](#) ([device_t](#) id, const [extio_settings_t](#) *extio_settings)
Set EXTIO settings.
- [result_t XIMC_API get_extio_settings](#) ([device_t](#) id, [extio_settings_t](#) *extio_settings)
Read EXTIO settings.
- [result_t XIMC_API set_brake_settings](#) ([device_t](#) id, const [brake_settings_t](#) *brake_settings)
Set brake control settings.
- [result_t XIMC_API get_brake_settings](#) ([device_t](#) id, [brake_settings_t](#) *brake_settings)
Read brake control settings.
- [result_t XIMC_API set_control_settings](#) ([device_t](#) id, const [control_settings_t](#) *control_settings)
Set motor control settings.
- [result_t XIMC_API set_control_settings_calb](#) ([device_t](#) id, const [control_settings_calb_t](#) *control_settings_calb, const [calibration_t](#) *calibration)
Set motor control settings with user units.
- [result_t XIMC_API get_control_settings](#) ([device_t](#) id, [control_settings_t](#) *control_settings)
Read motor control settings.
- [result_t XIMC_API get_control_settings_calb](#) ([device_t](#) id, [control_settings_calb_t](#) *control_settings_calb, const [calibration_t](#) *calibration)
Read motor control settings with user units.
- [result_t XIMC_API set_joystick_settings](#) ([device_t](#) id, const [joystick_settings_t](#) *joystick_settings)
Set joystick position.
- [result_t XIMC_API get_joystick_settings](#) ([device_t](#) id, [joystick_settings_t](#) *joystick_settings)
Read joystick settings.
- [result_t XIMC_API set_ctp_settings](#) ([device_t](#) id, const [ctp_settings_t](#) *ctp_settings)
Set control position settings (used with stepper motor only).
- [result_t XIMC_API get_ctp_settings](#) ([device_t](#) id, [ctp_settings_t](#) *ctp_settings)
Read control position settings (used with stepper motor only).
- [result_t XIMC_API set_uart_settings](#) ([device_t](#) id, const [uart_settings_t](#) *uart_settings)
Set UART settings.
- [result_t XIMC_API get_uart_settings](#) ([device_t](#) id, [uart_settings_t](#) *uart_settings)
Read UART settings.
- [result_t XIMC_API set_network_settings](#) ([device_t](#) id, const [network_settings_t](#) *network_settings)
Set network settings.
- [result_t XIMC_API get_network_settings](#) ([device_t](#) id, [network_settings_t](#) *network_settings)
Read network settings.
- [result_t XIMC_API set_password_settings](#) ([device_t](#) id, const [password_settings_t](#) *password_settings)
Sets the password.
- [result_t XIMC_API get_password_settings](#) ([device_t](#) id, [password_settings_t](#) *password_settings)
Read the password.

- `result_t XIMC_API set_calibration_settings (device_t id, const calibration_settings_t *calibration_settings)`
Set calibration settings.
- `result_t XIMC_API get_calibration_settings (device_t id, calibration_settings_t *calibration_settings)`
Read calibration settings.
- `result_t XIMC_API set_controller_name (device_t id, const controller_name_t *controller_name)`
Write user's controller name and internal settings to the FRAM.
- `result_t XIMC_API get_controller_name (device_t id, controller_name_t *controller_name)`
Read user's controller name and internal settings from the FRAM.
- `result_t XIMC_API set_nonvolatile_memory (device_t id, const nonvolatile_memory_t *nonvolatile_memory)`
Write user data into the FRAM.
- `result_t XIMC_API get_nonvolatile_memory (device_t id, nonvolatile_memory_t *nonvolatile_memory)`
Read user data from FRAM.
- `result_t XIMC_API set_emf_settings (device_t id, const emf_settings_t *emf_settings)`
Set electromechanical coefficients.
- `result_t XIMC_API get_emf_settings (device_t id, emf_settings_t *emf_settings)`
Read electromechanical settings.
- `result_t XIMC_API set_engine_advanced_setup (device_t id, const engine_advanced_setup_t *engine_advanced_setup)`
Set engine advanced settings.
- `result_t XIMC_API get_engine_advanced_setup (device_t id, engine_advanced_setup_t *engine_advanced_setup)`
Read engine advanced settings.
- `result_t XIMC_API set_extended_settings (device_t id, const extended_settings_t *extended_settings)`
Set extended settings.
- `result_t XIMC_API get_extended_settings (device_t id, extended_settings_t *extended_settings)`
Read extended settings.

Group of commands movement control

- `result_t XIMC_API command_stop (device_t id)`
Immediately stops the engine, moves it to the STOP state, and sets switches to BREAK mode (windings are short-circuited).
- `result_t XIMC_API command_power_off (device_t id)`
Immediately power off the motor regardless its state.
- `result_t XIMC_API command_move (device_t id, int Position, int uPosition)`
Move to position.
- `result_t XIMC_API command_move_calb (device_t id, float Position, const calibration_t *calibration)`
Move to position using user units.
- `result_t XIMC_API command_movr (device_t id, int DeltaPosition, int uDeltaPosition)`
Shift by a set offset.
- `result_t XIMC_API command_movr_calb (device_t id, float DeltaPosition, const calibration_t *calibration)`
Shift by a set offset using user units.
- `result_t XIMC_API command_home (device_t id)`
Moving to home position.
- `result_t XIMC_API command_left (device_t id)`
Start continuous moving to the left.
- `result_t XIMC_API command_right (device_t id)`
Start continuous moving to the right.
- `result_t XIMC_API command_loft (device_t id)`
Upon receiving the command "loft", the engine is shifted from the current position to a distance Antiplay defined in engine settings.
- `result_t XIMC_API command_sstp (device_t id)`
Soft stop the engine.

- [result_t XIMC_API get_position](#) ([device_t](#) id, [get_position_t](#) *the_get_position)
Reads the value position in steps and microsteps for stepper motor and encoder steps for all engines.
- [result_t XIMC_API get_position_calb](#) ([device_t](#) id, [get_position_calb_t](#) *the_get_position_calb, const [calibration_t](#) *calibration)
Reads position value in user units for stepper motor and encoder steps for all engines.
- [result_t XIMC_API set_position](#) ([device_t](#) id, const [set_position_t](#) *the_set_position)
Sets position in steps and microsteps for stepper motor.
- [result_t XIMC_API set_position_calb](#) ([device_t](#) id, const [set_position_calb_t](#) *the_set_position_calb, const [calibration_t](#) *calibration)
Sets any position value and encoder value of all engines.
- [result_t XIMC_API command_zero](#) ([device_t](#) id)
Sets the current position to 0.

Group of save settings and load settings commands

- [result_t XIMC_API command_save_settings](#) ([device_t](#) id)
Save all settings from the controller's RAM to the controller's flash memory, replacing previous data in the flash memory.
- [result_t XIMC_API command_read_settings](#) ([device_t](#) id)
Read all settings from the controller's flash memory to the controller's RAM, replacing previous data in the RAM.
- [result_t XIMC_API command_save_robust_settings](#) ([device_t](#) id)
Save important settings (calibration coefficients, etc.) from the controller's RAM to the controller's flash memory, replacing previous data in the flash memory.
- [result_t XIMC_API command_read_robust_settings](#) ([device_t](#) id)
Read important settings (calibration coefficients, etc.) from the controller's flash memory to the controller's RAM, replacing previous data in the RAM.
- [result_t XIMC_API command_eesave_settings](#) ([device_t](#) id)
Save settings from the controller's RAM to the stage's EEPROM.
- [result_t XIMC_API command_eeread_settings](#) ([device_t](#) id)
Read settings from the stage's EEPROM to the controller's RAM.
- [result_t XIMC_API command_start_measurements](#) ([device_t](#) id)
Start measurements and buffering of speed and the speed error (target speed minus real speed).
- [result_t XIMC_API get_measurements](#) ([device_t](#) id, [measurements_t](#) *measurements)
A command to read the data buffer to build a speed graph and a speed error graph.
- [result_t XIMC_API get_chart_data](#) ([device_t](#) id, [chart_data_t](#) *chart_data)
Return device electrical parameters, useful for charts.
- [result_t XIMC_API get_serial_number](#) ([device_t](#) id, unsigned int *SerialNumber)
Read device serial number.
- [result_t XIMC_API get_firmware_version](#) ([device_t](#) id, unsigned int *Major, unsigned int *Minor, unsigned int *Release)
Read the controller's firmware version.
- [result_t XIMC_API service_command_updf](#) ([device_t](#) id)
The command switches the controller to update the firmware state.

Service commands

- [result_t XIMC_API set_serial_number](#) ([device_t](#) id, const [serial_number_t](#) *serial_number)
Write device serial number and hardware version to the controller's flash memory.
- [result_t XIMC_API get_analog_data](#) ([device_t](#) id, [analog_data_t](#) *analog_data)
Read the analog data structure that contains raw analog data from the embedded ADC.
- [result_t XIMC_API get_debug_read](#) ([device_t](#) id, [debug_read_t](#) *debug_read)
Read data from firmware for debug purpose.
- [result_t XIMC_API set_debug_write](#) ([device_t](#) id, const [debug_write_t](#) *debug_write)
Write data to firmware for debug purpose.

A group of EEPROM commands

- `result_t XIMC_API set_stage_name (device_t id, const stage_name_t *stage_name)`
Write the user's stage name to EEPROM.
- `result_t XIMC_API get_stage_name (device_t id, stage_name_t *stage_name)`
Read the user's stage name from the EEPROM.
- `result_t XIMC_API set_stage_information (device_t id, const stage_information_t *stage_↵
information)`
Deprecated.
- `result_t XIMC_API get_stage_information (device_t id, stage_information_t *stage_information)`
Deprecated.
- `result_t XIMC_API set_stage_settings (device_t id, const stage_settings_t *stage_settings)`
Deprecated.
- `result_t XIMC_API get_stage_settings (device_t id, stage_settings_t *stage_settings)`
Deprecated.
- `result_t XIMC_API set_motor_information (device_t id, const motor_information_t *motor_↵
information)`
Deprecated.
- `result_t XIMC_API get_motor_information (device_t id, motor_information_t *motor_information)`
Deprecated.
- `result_t XIMC_API set_motor_settings (device_t id, const motor_settings_t *motor_settings)`
Deprecated.
- `result_t XIMC_API get_motor_settings (device_t id, motor_settings_t *motor_settings)`
Deprecated.
- `result_t XIMC_API set_encoder_information (device_t id, const encoder_information_t *encoder_↵
information)`
Deprecated.
- `result_t XIMC_API get_encoder_information (device_t id, encoder_information_t *encoder_↵
information)`
Deprecated.
- `result_t XIMC_API set_encoder_settings (device_t id, const encoder_settings_t *encoder_settings)`
Deprecated.
- `result_t XIMC_API get_encoder_settings (device_t id, encoder_settings_t *encoder_settings)`
Deprecated.
- `result_t XIMC_API set_hallsensor_information (device_t id, const hallsensor_information_t
*hallsensor_information)`
Deprecated.
- `result_t XIMC_API get_hallsensor_information (device_t id, hallsensor_information_t *hallsensor_↵
information)`
Deprecated.
- `result_t XIMC_API set_hallsensor_settings (device_t id, const hallsensor_settings_t *hallsensor_↵
settings)`
Deprecated.
- `result_t XIMC_API get_hallsensor_settings (device_t id, hallsensor_settings_t *hallsensor_settings)`
Deprecated.
- `result_t XIMC_API set_gear_information (device_t id, const gear_information_t *gear_information)`
Deprecated.
- `result_t XIMC_API get_gear_information (device_t id, gear_information_t *gear_information)`
Deprecated.
- `result_t XIMC_API set_gear_settings (device_t id, const gear_settings_t *gear_settings)`
Deprecated.
- `result_t XIMC_API get_gear_settings (device_t id, gear_settings_t *gear_settings)`
Deprecated.
- `result_t XIMC_API set_accessories_settings (device_t id, const accessories_settings_t *accessories_↵
_settings)`
Deprecated.
- `result_t XIMC_API get_accessories_settings (device_t id, accessories_settings_t *accessories_↵
settings)`

- Deprecated.*

 - [result_t XIMC_API get_bootloader_version](#) ([device_t](#) id, unsigned int *Major, unsigned int *Minor, unsigned int *Release)
 - Read the controller's bootloader version.*
 - [result_t XIMC_API get_init_random](#) ([device_t](#) id, [init_random_t](#) *init_random)
 - Read a random number from the controller.*
 - [result_t XIMC_API get_globally_unique_identifier](#) ([device_t](#) id, [globally_unique_identifier_t](#) *globally_unique_identifier)
 - This value is unique to each individual device, but is not a random value.*
 - [result_t XIMC_API goto_firmware](#) ([device_t](#) id, [uint8_t](#) *ret)
 - Reboot to firmware*
 - [result_t XIMC_API has_firmware](#) (const char *uri, [uint8_t](#) *ret)
 - Check for firmware on device*
 - [result_t XIMC_API command_update_firmware](#) (const char *uri, const [uint8_t](#) *data, [uint32_t](#) data_size)
 - Update firmware.*
 - [result_t XIMC_API write_key](#) (const char *uri, [uint8_t](#) *key)
 - Write controller key.*
 - [result_t XIMC_API command_reset](#) ([device_t](#) id)
 - Reset controller.*
 - [result_t XIMC_API command_clear_fram](#) ([device_t](#) id)
 - Clear controller FRAM.*

Logging level

- `#define LOGLEVEL_ERROR 0x01`
Logging level - error
- `#define LOGLEVEL_WARNING 0x02`
Logging level - warning
- `#define LOGLEVEL_INFO 0x03`
Logging level - info
- `#define LOGLEVEL_DEBUG 0x04`
Logging level - debug
- `typedef struct calibration_t` [calibration_t](#)
Calibration structure
- `typedef struct device_network_information_t` [device_network_information_t](#)
Device network information structure.

Boards and drivers control

Functions for searching and opening/closing devices

- `typedef char * pchar`
Nevermind
- `typedef void(XIMC_CALLCONV * logging_callback_t)` (int loglevel, const [wchar_t](#) *message, void *user_data)
Logging callback prototype
- [device_t](#) [XIMC_API open_device](#) (const char *uri)
Open a device with OS uri and return identifier of the device which can be used in calls.
- [result_t](#) [XIMC_API close_device](#) ([device_t](#) *id)
Close specified device

- [result_t XIMC_API set_correction_table](#) ([device_t](#) id, const char *namefile)
Command of loading a correction table from a text file.
- [result_t XIMC_API probe_device](#) (const char *uri)
Check if a device with OS uri uri is XIMC device.
- [device_enumeration_t XIMC_API enumerate_devices](#) (int enumerate_flags, const char *hints)
Search and list of available devices.
- [result_t XIMC_API free_enumerate_devices](#) ([device_enumeration_t](#) device_enumeration)
Free memory returned by enumerate_devices.
- [int XIMC_API get_device_count](#) ([device_enumeration_t](#) device_enumeration)
Get device count.
- [pchar XIMC_API get_device_name](#) ([device_enumeration_t](#) device_enumeration, int device_index)
Get device name from the device enumeration.
- [result_t XIMC_API get_enumerate_device_serial](#) ([device_enumeration_t](#) device_enumeration, int device_index, [uint32_t](#) *serial)
Get device serial number from the device enumeration.
- [result_t XIMC_API get_enumerate_device_information](#) ([device_enumeration_t](#) device_enumeration, int device_index, [device_information_t](#) *device_information)
Get device information from the device enumeration.
- [result_t XIMC_API get_enumerate_device_controller_name](#) ([device_enumeration_t](#) device_enumeration, int device_index, [controller_name_t](#) *controller_name)
Get controller name from the device enumeration.
- [result_t XIMC_API get_enumerate_device_stage_name](#) ([device_enumeration_t](#) device_enumeration, int device_index, [stage_name_t](#) *stage_name)
Get stage name from the device enumeration.
- [result_t XIMC_API get_enumerate_device_network_information](#) ([device_enumeration_t](#) device_enumeration, int device_index, [device_network_information_t](#) *device_network_information)
Get device network information from the device enumeration.
- [result_t XIMC_API reset_locks](#) ()
Resets the error of incorrect data transmission.
- [void XIMC_API msec_sleep](#) (unsigned int msec)
Sleeps for a specified amount of time
- [void XIMC_API ximc_version](#) (char *version)
Returns a library version
- [void XIMC_API logging_callback_stderr_wide](#) (int loglevel, const [wchar_t](#) *message, void *user_data)
Simple callback for logging to stderr in wide chars
- [void XIMC_API logging_callback_stderr_narrow](#) (int loglevel, const [wchar_t](#) *message, void *user_data)
Simple callback for logging to stderr in narrow (single byte) chars
- [void XIMC_API set_logging_callback](#) ([logging_callback_t](#) logging_callback, void *user_data)
Sets a logging callback.
- [result_t XIMC_API get_status](#) ([device_t](#) id, [status_t](#) *status)
Return device state.
- [result_t XIMC_API get_status_calb](#) ([device_t](#) id, [status_calb_t](#) *status, const [calibration_t](#) *calibration)
Return device state.
- [result_t XIMC_API get_device_information](#) ([device_t](#) id, [device_information_t](#) *device_information)
Return device information.
- [result_t XIMC_API command_wait_for_stop](#) ([device_t](#) id, [uint32_t](#) refresh_interval_ms)
Wait for stop
- [result_t XIMC_API command_homezero](#) ([device_t](#) id)
Make home command, wait until it is finished and make zero command.

6.1.1 Detailed Description

Header file for libximc library

6.1.2 Macro Definition Documentation

6.1.2.1 ALARM_ON_DRIVER_OVERHEATING

```
#define ALARM_ON_DRIVER_OVERHEATING 0x01
```

If this flag is set, enter the alarm state on the driver overheat signal.

6.1.2.2 BACK_EMF_INDUCTANCE_AUTO

```
#define BACK_EMF_INDUCTANCE_AUTO 0x01
```

Flag of auto-detection of inductance of windings of the engine.

6.1.2.3 BACK_EMF_KM_AUTO

```
#define BACK_EMF_KM_AUTO 0x04
```

Flag of auto-detection of electromechanical coefficient of the engine.

6.1.2.4 BACK_EMF_RESISTANCE_AUTO

```
#define BACK_EMF_RESISTANCE_AUTO 0x02
```

Flag of auto-detection of resistance of windings of the engine.

6.1.2.5 BORDER_IS_ENCODER

```
#define BORDER_IS_ENCODER 0x01
```

Borders are fixed by predetermined encoder values, if set; borders are placed on limit switches, if not set.

6.1.2.6 BORDER_STOP_LEFT

```
#define BORDER_STOP_LEFT 0x02
```

The motor should stop on the left border.

6.1.2.7 BORDER_STOP_RIGHT

```
#define BORDER_STOP_RIGHT 0x04
```

Motor should stop on right border.

6.1.2.8 BORDERS_SWAP_MISSET_DETECTION

```
#define BORDERS_SWAP_MISSET_DETECTION 0x08
```

Motor should stop on both borders.

Need to save the motor when wrong border settings is set

6.1.2.9 BRAKE_ENABLED

```
#define BRAKE_ENABLED 0x01
```

Brake control is enabled if this flag is set.

6.1.2.10 BRAKE_ENG_PWROFF

```
#define BRAKE_ENG_PWROFF 0x02
```

Brake turns the stepper motor power off if this flag is set.

6.1.2.11 BRAKING_OVERVOLTAGE_PROTECTION

```
#define BRAKING_OVERVOLTAGE_PROTECTION 0x20
```

If this flag is set, the firmware will close ground switches of H-bridge to disconnect motor from power circuit on overvoltage.

6.1.2.12 CONTROL_BTN_LEFT_PUSHED_OPEN

```
#define CONTROL_BTN_LEFT_PUSHED_OPEN 0x04
```

Pushed left button corresponds to the open contact if this flag is set.

6.1.2.13 CONTROL_BTN_RIGHT_PUSHED_OPEN

```
#define CONTROL_BTN_RIGHT_PUSHED_OPEN 0x08
```

Pushed right button corresponds to open contact if this flag is set.

6.1.2.14 CONTROL_MODE_BITS

```
#define CONTROL_MODE_BITS 0x03
```

Bits to control the engine by joystick or buttons.

6.1.2.15 CONTROL_MODE_JOY

```
#define CONTROL_MODE_JOY 0x01
```

Control by joystick.

6.1.2.16 CONTROL_MODE_LR

```
#define CONTROL_MODE_LR 0x02
```

Control by left/right buttons.

6.1.2.17 CONTROL_MODE_OFF

```
#define CONTROL_MODE_OFF 0x00
```

Control is disabled.

6.1.2.18 CTP_ALARM_ON_ERROR

```
#define CTP_ALARM_ON_ERROR 0x04
```

Set ALARM on mismatch if the flag is set.

6.1.2.19 CTP_BASE

```
#define CTP_BASE 0x02
```

The position control is based on the revolution sensor if this flag is set; otherwise, it is based on the encoder.

6.1.2.20 CTP_ENABLED

```
#define CTP_ENABLED 0x01
```

The position control is enabled if the flag is set.

6.1.2.21 CTP_ERROR_CORRECTION

```
#define CTP_ERROR_CORRECTION 0x10
```

Correct errors that appear when slippage occurs if the flag is set.

It works only with the encoder. Incompatible with the flag CTP_ALARM_ON_ERROR.

6.1.2.22 DRIVER_TYPE_EXTERNAL

```
#define DRIVER_TYPE_EXTERNAL 0x03
```

External driver.

6.1.2.23 DRIVER_TYPE_INTEGRATE

```
#define DRIVER_TYPE_INTEGRATE 0x02
```

Driver with integrated IC.

6.1.2.24 EEPROM_PRECEDENCE

```
#define EEPROM_PRECEDENCE 0x01
```

If the flag is set, settings from external EEPROM override controller settings.

6.1.2.25 ENC_STATE_ABSENT

```
#define ENC_STATE_ABSENT 0x00
```

Encoder is absent.

6.1.2.26 ENC_STATE_MALFUNC

```
#define ENC_STATE_MALFUNC 0x02
```

Encoder is connected and malfunctioning.

6.1.2.27 ENC_STATE_OK

```
#define ENC_STATE_OK 0x04
```

Encoder is connected and working properly.

6.1.2.28 ENC_STATE_REVERS

```
#define ENC_STATE_REVERS 0x03
```

Encoder is connected and operational but counts in other direction.

6.1.2.29 ENC_STATE_UNKNOWN

```
#define ENC_STATE_UNKNOWN 0x01
```

Encoder state is unknown.

6.1.2.30 ENDER_SW1_ACTIVE_LOW

```
#define ENDER_SW1_ACTIVE_LOW 0x02
```

1 - Limit switch connected to pin SW1 is triggered by a low level on pin.

6.1.2.31 ENDER_SW2_ACTIVE_LOW

```
#define ENDER_SW2_ACTIVE_LOW 0x04
```

1 - Limit switch connected to pin SW2 is triggered by a low level on pin.

6.1.2.32 ENDER_SWAP

```
#define ENDER_SWAP 0x01
```

First limit switch on the right side, if set; otherwise on the left side.

6.1.2.33 ENGINE_ACCEL_ON

```
#define ENGINE_ACCEL_ON 0x10
```

Acceleration enable flag.

If it set, motion begins with acceleration and ends with deceleration.

6.1.2.34 ENGINE_ANTIPLAY

```
#define ENGINE_ANTIPLAY 0x08
```

Play compensation flag.

If it is set, the engine makes backlash (play) compensation and reaches the predetermined position accurately at low speed.

6.1.2.35 ENGINE_CURRENT_AS_RMS

```
#define ENGINE_CURRENT_AS_RMS 0x02
```

Engine current meaning flag.

If the flag is unset, then the engine's current value is interpreted as the maximum amplitude value. If the flag is set, then the engine current value is interpreted as the root-mean-square current value (for stepper) or as the current value calculated from the maximum heat dissipation (BLDC).

6.1.2.36 ENGINE_LIMIT_CURR

```
#define ENGINE_LIMIT_CURR 0x40
```

Maximum motor current limit enable flag (is only used with DC motor).

6.1.2.37 ENGINE_LIMIT_RPM

```
#define ENGINE_LIMIT_RPM 0x80
```

Maximum motor speed limit enable flag.

6.1.2.38 ENGINE_LIMIT_VOLT

```
#define ENGINE_LIMIT_VOLT 0x20
```

Maximum motor voltage limit enable flag (is only used with DC motor).

6.1.2.39 ENGINE_MAX_SPEED

```
#define ENGINE_MAX_SPEED 0x04
```

Max speed flag.

If it is set, the engine uses the maximum speed achievable with the present engine settings as its nominal speed.

6.1.2.40 ENGINE_REVERSE

```
#define ENGINE_REVERSE 0x01
```

Reverse flag.

It determines motor shaft rotation direction that corresponds to feedback counts increasing. If not set (default), motor shaft rotation direction under positive voltage corresponds to feedback counts increasing and vice versa. Change it if you see that positive directions on motor and feedback are opposite.

6.1.2.41 ENGINE_TYPE_2DC

```
#define ENGINE_TYPE_2DC 0x02
```

2 DC motors.

6.1.2.42 ENGINE_TYPE_BRUSHLESS

```
#define ENGINE_TYPE_BRUSHLESS 0x05
```

Brushless motor.

6.1.2.43 ENGINE_TYPE_DC

```
#define ENGINE_TYPE_DC 0x01
```

DC motor.

6.1.2.44 ENGINE_TYPE_NONE

```
#define ENGINE_TYPE_NONE 0x00
```

A value that shouldn't be used.

6.1.2.45 ENGINE_TYPE_STEP

```
#define ENGINE_TYPE_STEP 0x03
```

Step motor.

6.1.2.46 ENGINE_TYPE_TEST

```
#define ENGINE_TYPE_TEST 0x04
```

Duty cycle are fixed.

Used only manufacturer.

6.1.2.47 ENUMERATE_PROBE

```
#define ENUMERATE_PROBE 0x01
```

Check if a device with an OS name is a XIMC device.

Be careful with this flag because it sends some data to the device.

6.1.2.48 EXTIO_SETUP_INVERT

```
#define EXTIO_SETUP_INVERT 0x02
```

Interpret EXTIO state inverted if the flag is set.

A falling front is treated as an input event and a low logic level as an active state.

6.1.2.49 EXTIO_SETUP_MODE_IN_ALARM

```
#define EXTIO_SETUP_MODE_IN_ALARM 0x05
```

Set Alarm when the signal goes to the active state.

6.1.2.50 EXTIO_SETUP_MODE_IN_BITS

```
#define EXTIO_SETUP_MODE_IN_BITS 0x0F
```

Bits of the behavior selector when the signal on input goes to the active state.

6.1.2.51 EXTIO_SETUP_MODE_IN_HOME

```
#define EXTIO_SETUP_MODE_IN_HOME 0x04
```

Issue HOME command.

6.1.2.52 EXTIO_SETUP_MODE_IN_MOVR

```
#define EXTIO_SETUP_MODE_IN_MOVR 0x03
```

Issue MOVR command with last used settings.

6.1.2.53 EXTIO_SETUP_MODE_IN_NOP

```
#define EXTIO_SETUP_MODE_IN_NOP 0x00
```

Do nothing.

6.1.2.54 EXTIO_SETUP_MODE_IN_PWOF

```
#define EXTIO_SETUP_MODE_IN_PWOF 0x02
```

Issue PWOF command, powering off all engine windings.

6.1.2.55 EXTIO_SETUP_MODE_IN_STOP

```
#define EXTIO_SETUP_MODE_IN_STOP 0x01
```

Issue STOP command, ceasing the engine movement.

6.1.2.56 EXTIO_SETUP_MODE_OUT_ALARM

```
#define EXTIO_SETUP_MODE_OUT_ALARM 0x30
```

EXTIO pin stays active during the alarm state.

6.1.2.57 EXTIO_SETUP_MODE_OUT_BITS

```
#define EXTIO_SETUP_MODE_OUT_BITS 0xF0
```

Bits of the output behavior selection.

6.1.2.58 EXTIO_SETUP_MODE_OUT_MOTOR_ON

```
#define EXTIO_SETUP_MODE_OUT_MOTOR_ON 0x40
```

EXTIO pin stays active when windings are powered.

6.1.2.59 EXTIO_SETUP_MODE_OUT_MOVING

```
#define EXTIO_SETUP_MODE_OUT_MOVING 0x20
```

EXTIO pin stays active during moving state.

6.1.2.60 EXTIO_SETUP_MODE_OUT_OFF

```
#define EXTIO_SETUP_MODE_OUT_OFF 0x00
```

EXTIO pin always set in inactive state.

6.1.2.61 EXTIO_SETUP_MODE_OUT_ON

```
#define EXTIO_SETUP_MODE_OUT_ON 0x10
```

EXTIO pin always set in active state.

6.1.2.62 EXTIO_SETUP_OUTPUT

```
#define EXTIO_SETUP_OUTPUT 0x01
```

EXTIO works as output if the flag is set, works as input otherwise.

6.1.2.63 FEEDBACK_EMF

```
#define FEEDBACK_EMF 0x04
```

Feedback by EMF.

6.1.2.64 FEEDBACK_ENC_ADAPTIVE_HOLDING

```
#define FEEDBACK_ENC_ADAPTIVE_HOLDING 0x02
```

Enables the adaptive holding algorithm.

6.1.2.65 FEEDBACK_ENC_FILTER_BITS

```
#define FEEDBACK_ENC_FILTER_BITS 0x30
```

Bits responsible for setting the internal filter of the encoder signal.

6.1.2.66 FEEDBACK_ENC_FILTER_MEDIUM

```
#define FEEDBACK_ENC_FILTER_MEDIUM 0x20
```

Medium noise filtering: the maximum encoder signal frequency is 1 MHz.

6.1.2.67 FEEDBACK_ENC_FILTER_NONE

```
#define FEEDBACK_ENC_FILTER_NONE 0x00
```

Disable the internal filter of the encoder signal.

6.1.2.68 FEEDBACK_ENC_FILTER_STRONG

```
#define FEEDBACK_ENC_FILTER_STRONG 0x30
```

Strong noise filtering: the maximum encoder signal frequency is 300 kHz.

6.1.2.69 FEEDBACK_ENC_FILTER_WEAK

```
#define FEEDBACK_ENC_FILTER_WEAK 0x10
```

Weak noise filtering: the maximum encoder signal frequency is 3 MHz.

6.1.2.70 FEEDBACK_ENC_REVERSE

```
#define FEEDBACK_ENC_REVERSE 0x01
```

Reverse count of encoder.

6.1.2.71 FEEDBACK_ENC_TYPE_AUTO

```
#define FEEDBACK_ENC_TYPE_AUTO 0x00
```

Auto detect encoder type.

6.1.2.72 FEEDBACK_ENC_TYPE_BITS

```
#define FEEDBACK_ENC_TYPE_BITS 0xC0
```

Bits of the encoder type.

6.1.2.73 FEEDBACK_ENC_TYPE_DIFFERENTIAL

```
#define FEEDBACK_ENC_TYPE_DIFFERENTIAL 0x80
```

Differential encoder.

6.1.2.74 FEEDBACK_ENC_TYPE_SINGLE_ENDED

```
#define FEEDBACK_ENC_TYPE_SINGLE_ENDED 0x40
```

Single-ended encoder.

6.1.2.75 FEEDBACK_ENCODER

```
#define FEEDBACK_ENCODER 0x01
```

Feedback by encoder.

6.1.2.76 FEEDBACK_ENCODER_MEDIATED

```
#define FEEDBACK_ENCODER_MEDIATED 0x06
```

Feedback by encoder mediated by mechanical transmission (for example leadscrew).

6.1.2.77 FEEDBACK_NONE

```
#define FEEDBACK_NONE 0x05
```

Feedback is absent.

6.1.2.78 H_BRIDGE_ALERT

```
#define H_BRIDGE_ALERT 0x04
```

If this flag is set, then turn off the power unit with a signal problem in one of the transistor bridge.

6.1.2.79 HOME_DIR_FIRST

```
#define HOME_DIR_FIRST 0x001
```

The flag defines the direction of the 1st motion after execution of the home command.

The direction is to the right if the flag is set, and to the left otherwise.

6.1.2.80 HOME_DIR_SECOND

```
#define HOME_DIR_SECOND 0x002
```

The flag defines the direction of the 2nd motion.

The direction is to the right if the flag is set, and to the left otherwise.

6.1.2.81 HOME_HALF_MV

```
#define HOME_HALF_MV 0x008
```

If the flag is set, the stop signals are ignored during the first half-turn of the second movement.

6.1.2.82 HOME_MV_SEC_EN

```
#define HOME_MV_SEC_EN 0x004
```

Use the second phase of calibration to the home position, if set; otherwise the second phase is skipped.

6.1.2.83 HOME_STOP_FIRST_BITS

```
#define HOME_STOP_FIRST_BITS 0x030
```

Bits of the first stop selector.

6.1.2.84 HOME_STOP_FIRST_LIM

```
#define HOME_STOP_FIRST_LIM 0x030
```

First motion stops by limit switch.

6.1.2.85 HOME_STOP_FIRST_REV

```
#define HOME_STOP_FIRST_REV 0x010
```

First motion stops by revolution sensor.

6.1.2.86 HOME_STOP_FIRST_SYN

```
#define HOME_STOP_FIRST_SYN 0x020
```

First motion stops by synchronization input.

6.1.2.87 HOME_STOP_SECOND_BITS

```
#define HOME_STOP_SECOND_BITS 0x0C0
```

Bits of the second stop selector.

6.1.2.88 HOME_STOP_SECOND_LIM

```
#define HOME_STOP_SECOND_LIM 0x0C0
```

Second motion stops by limit switch.

6.1.2.89 HOME_STOP_SECOND_REV

```
#define HOME_STOP_SECOND_REV 0x040
```

Second motion stops by revolution sensor.

6.1.2.90 HOME_STOP_SECOND_SYN

```
#define HOME_STOP_SECOND_SYN 0x080
```

Second motion stops by synchronization input.

6.1.2.91 HOME_USE_FAST

```
#define HOME_USE_FAST 0x100
```

Use the fast algorithm of calibration to the home position, if set; otherwise the traditional algorithm.

6.1.2.92 JOY_REVERSE

```
#define JOY_REVERSE 0x01
```

Joystick action is reversed.

The joystick deviation to the upper values corresponds to negative speed and vice versa.

6.1.2.93 LOW_UPWR_PROTECTION

```
#define LOW_UPWR_PROTECTION 0x02
```

If this flag is set, turn off the motor when the voltage is lower than LowUpwrOff.

6.1.2.94 MICROSTEP_MODE_FRAC_128

```
#define MICROSTEP_MODE_FRAC_128 0x08
```

1/128-step mode.

6.1.2.95 MICROSTEP_MODE_FRAC_16

```
#define MICROSTEP_MODE_FRAC_16 0x05
```

1/16-step mode.

6.1.2.96 MICROSTEP_MODE_FRAC_2

```
#define MICROSTEP_MODE_FRAC_2 0x02
```

1/2-step mode.

6.1.2.97 MICROSTEP_MODE_FRAC_256

```
#define MICROSTEP_MODE_FRAC_256 0x09
```

1/256-step mode.

6.1.2.98 MICROSTEP_MODE_FRAC_32

```
#define MICROSTEP_MODE_FRAC_32 0x06
```

1/32-step mode.

6.1.2.99 MICROSTEP_MODE_FRAC_4

```
#define MICROSTEP_MODE_FRAC_4 0x03
```

1/4-step mode.

6.1.2.100 MICROSTEP_MODE_FRAC_64

```
#define MICROSTEP_MODE_FRAC_64 0x07
```

1/64-step mode.

6.1.2.101 MICROSTEP_MODE_FRAC_8

```
#define MICROSTEP_MODE_FRAC_8 0x04
```

1/8-step mode.

6.1.2.102 MICROSTEP_MODE_FULL

```
#define MICROSTEP_MODE_FULL 0x01
```

Full step mode.

6.1.2.103 MOVE_STATE_ANTIPLAY

```
#define MOVE_STATE_ANTIPLAY 0x04
```

Motor is playing compensation, if flag set.

6.1.2.104 MOVE_STATE_MOVING

```
#define MOVE_STATE_MOVING 0x01
```

This flag indicates that the controller is trying to move the motor.

Don't use this flag to wait for the completion of the movement command. Use the MVCMD_RUNNING flag from the MvCmdSts field instead.

6.1.2.105 MOVE_STATE_TARGET_SPEED

```
#define MOVE_STATE_TARGET_SPEED 0x02
```

Target speed is reached, if flag set.

6.1.2.106 MVCMD_ERROR

```
#define MVCMD_ERROR 0x40
```

Finish state (1 - move command has finished with an error, 0 - move command has finished correctly).

This flag makes sense when MVCMD_RUNNING signals movement completion.

6.1.2.107 MVCMD_HOME

```
#define MVCMD_HOME 0x06
```

Command home.

6.1.2.108 MVCMD_LEFT

```
#define MVCMD_LEFT 0x03
```

Command left.

6.1.2.109 MVCMD_LOFT

```
#define MVCMD_LOFT 0x07
```

Command loft.

6.1.2.110 MVCMD_MOVE

```
#define MVCMD_MOVE 0x01
```

Command move.

6.1.2.111 MVCMD_MOVR

```
#define MVCMD_MOVR 0x02
```

Command movr.

6.1.2.112 MVCMD_NAME_BITS

```
#define MVCMD_NAME_BITS 0x3F
```

Move command bit mask.

6.1.2.113 MVCMD_RIGHT

```
#define MVCMD_RIGHT 0x04
```

Command rigt.

6.1.2.114 MVCMD_RUNNING

```
#define MVCMD_RUNNING 0x80
```

Move command state (0 - move command has finished, 1 - move command is being executed).

6.1.2.115 MVCMD_SSTP

```
#define MVCMD_SSTP 0x08
```

Command soft stop.

6.1.2.116 MVCMD_STOP

```
#define MVCMD_STOP 0x05
```

Command stop.

6.1.2.117 MVCMD_UKNWN

```
#define MVCMD_UKNWN 0x00
```

Unknown command.

6.1.2.118 POWER_OFF_ENABLED

```
#define POWER_OFF_ENABLED 0x02
```

Power off is enabled after PowerOffDelay if this flag is set.

6.1.2.119 POWER_REDUCT_ENABLED

```
#define POWER_REDUCT_ENABLED 0x01
```

Current reduction is enabled after CurrReductDelay if this flag is set.

6.1.2.120 POWER_SMOOTH_CURRENT

```
#define POWER_SMOOTH_CURRENT 0x04
```

Current ramp-up/down are performed smoothly during current_set_time if this flag is set.

6.1.2.121 PWR_STATE_MAX

```
#define PWR_STATE_MAX 0x05
```

Motor windings are powered by the maximum current driver can provide at this voltage.

6.1.2.122 PWR_STATE_NORM

```
#define PWR_STATE_NORM 0x03
```

Motor windings are powered by nominal current.

6.1.2.123 PWR_STATE_OFF

```
#define PWR_STATE_OFF 0x01
```

Motor windings are disconnected from the driver.

6.1.2.124 PWR_STATE_REDUCT

```
#define PWR_STATE_REDUCT 0x04
```

Motor windings are powered by reduced current to lower power consumption.

6.1.2.125 PWR_STATE_UNKNOWN

```
#define PWR_STATE_UNKNOWN 0x00
```

Unknown state, should never happen.

6.1.2.126 REV_SENS_INV

```
#define REV_SENS_INV 0x08
```

Typically, the sensor is active when it is at 0, and inversion makes active at 1.

That is, if you do not invert, it is normal logic - 0 is the activation.

6.1.2.127 RPM_DIV_1000

```
#define RPM_DIV_1000 0x01
```

This flag indicates that the operating speed specified in the command is set in milliRPM.

Applicable only for ENCODER feedback mode and only for BLDC motors.

6.1.2.128 SETPOS_IGNORE_ENCODER

```
#define SETPOS_IGNORE_ENCODER 0x02
```

Will not reload encoder state if this flag is set.

6.1.2.129 SETPOS_IGNORE_POSITION

```
#define SETPOS_IGNORE_POSITION 0x01
```

Will not reload position in steps/microsteps if this flag is set.

6.1.2.130 STATE_ALARM

```
#define STATE_ALARM 0x0000040
```

The controller is in an alarm state, indicating that something dangerous has happened.

Most commands are ignored in this state. To reset the flag, a STOP command must be issued.

6.1.2.131 STATE_BORDERS_SWAP_MISSET

```
#define STATE_BORDERS_SWAP_MISSET 0x0008000
```

Engine stuck at the wrong edge.

6.1.2.132 STATE_BRAKE

```
#define STATE_BRAKE 0x0200
```

State of Brake pin.

Flag "1" - if the pin state brake is not powered (brake is clamped), "0" - if the pin state brake is powered (brake is unclamped).

6.1.2.133 STATE_BUTTON_LEFT

```
#define STATE_BUTTON_LEFT 0x0008
```

Button "left" state (1 if pressed).

6.1.2.134 STATE_BUTTON_RIGHT

```
#define STATE_BUTTON_RIGHT 0x0004
```

Button "right" state (1 if pressed).

6.1.2.135 STATE_CONTR

```
#define STATE_CONTR 0x000003F
```

Flags of controller states.

6.1.2.136 STATE_CONTROLLER_OVERHEAT

```
#define STATE_CONTROLLER_OVERHEAT 0x0000200
```

Controller overheat.

6.1.2.137 STATE_CTP_ERROR

```
#define STATE_CTP_ERROR 0x0000080
```

Control position error (is only used with stepper motor).

The flag is set when the encoder position and step position are too far apart.

6.1.2.138 STATE_DIG_SIGNAL

```
#define STATE_DIG_SIGNAL 0xFFFF
```

Flags of digital signals.

6.1.2.139 STATE_EEPROM_CONNECTED

```
#define STATE_EEPROM_CONNECTED 0x0000010
```

EEPROM with settings is connected.

The built-in stage profile is uploaded from the EEPROM memory chip if the EEPROM_PRECEDENCE flag is set, allowing you to connect various stages to the controller with automatic setup.

6.1.2.140 STATE_ENC_A

```
#define STATE_ENC_A 0x2000
```

State of encoder A pin.

6.1.2.141 STATE_ENC_B

```
#define STATE_ENC_B 0x4000
```

State of encoder B pin.

6.1.2.142 STATE_ENGINE_RESPONSE_ERROR

```
#define STATE_ENGINE_RESPONSE_ERROR 0x0800000
```

Error response of the engine control action.

Motor control algorithm failure means that it can't make the correct decisions with the feedback data it receives. A single failure may be caused by a mechanical problem. A repeating failure can be caused by incorrect motor settings.

6.1.2.143 STATE_ERRC

```
#define STATE_ERRC 0x0000001
```

Command error encountered.

The command received is not in the list of controller known commands. The most possible reason is the outdated firmware.

6.1.2.144 STATE_ERRD

```
#define STATE_ERRD 0x0000002
```

Data integrity error encountered.

The data inside the command and its CRC code do not correspond. Therefore, the data can't be considered valid. This error may be caused by EMI in the UART/RS232 interface.

6.1.2.145 STATE_ERRV

```
#define STATE_ERRV 0x0000004
```

Value error encountered.

The values in the command can't be applied without correction because they fall outside the valid range. Corrected values were used instead of the original ones.

6.1.2.146 STATE_EXTIO_ALARM

```
#define STATE_EXTIO_ALARM 0x1000000
```

The error is caused by the external EXTIO input signal.

6.1.2.147 STATE_GPIO_LEVEL

```
#define STATE_GPIO_LEVEL 0x0020
```

State of external GPIO pin.

6.1.2.148 STATE_GPIO_PINOUT

```
#define STATE_GPIO_PINOUT 0x0010
```

External GPIO works as out if the flag is set; otherwise, it works as in.

6.1.2.149 STATE_IS_HOMED

```
#define STATE_IS_HOMED 0x0000020
```

Calibration performed.

This means that the relative position scale is calibrated against a hardware absolute position sensor, like a limit switch. Drops after loss of calibration, like harsh stops and possibly skipped steps.

6.1.2.150 STATE_LEFT_EDGE

```
#define STATE_LEFT_EDGE 0x0002
```

Engine stuck at the left edge.

6.1.2.151 STATE_LOW_USB_VOLTAGE

```
#define STATE_LOW_USB_VOLTAGE 0x0002000
```

Deprecated.

USB voltage is insufficient for normal operation. This field is left for compatibility. Software should not rely on the value of this field.

6.1.2.152 STATE_OVERLOAD_POWER_CURRENT

```
#define STATE_OVERLOAD_POWER_CURRENT 0x0000800
```

Power current exceeds safe limit.

6.1.2.153 STATE_OVERLOAD_POWER_VOLTAGE

```
#define STATE_OVERLOAD_POWER_VOLTAGE 0x0000400
```

Power voltage exceeds safe limit.

6.1.2.154 STATE_OVERLOAD_USB_CURRENT

```
#define STATE_OVERLOAD_USB_CURRENT 0x0004000
```

Deprecated.

USB current exceeds safe limit. This field is left for compatibility. Software should not rely on the value of this field.

6.1.2.155 STATE_OVERLOAD_USB_VOLTAGE

```
#define STATE_OVERLOAD_USB_VOLTAGE 0x0001000
```

Deprecated.

USB voltage exceeds safe limit. This field is left for compatibility. Software should not rely on the value of this field.

6.1.2.156 STATE_POWER_OVERHEAT

```
#define STATE_POWER_OVERHEAT 0x0000100
```

Power driver overheat.

Motor control is disabled until some cooldown occurs. This should not happen with boxed versions of the controller. This may happen with the bare-board version of the controller with a custom radiator. Redesign your radiator.

6.1.2.157 STATE_REV_SENSOR

```
#define STATE_REV_SENSOR 0x0400
```

State of Revolution sensor pin.

6.1.2.158 STATE_RIGHT_EDGE

```
#define STATE_RIGHT_EDGE 0x0001
```

Engine stuck at the right edge.

6.1.2.159 STATE_SECUR

```
#define STATE_SECUR 0x1B3FFC0
```

Security flags.

6.1.2.160 STATE_SYNC_INPUT

```
#define STATE_SYNC_INPUT 0x0800
```

State of Sync input pin.

6.1.2.161 STATE_SYNC_OUTPUT

```
#define STATE_SYNC_OUTPUT 0x1000
```

State of Sync output pin.

6.1.2.162 STATE_WINDING_RES_MISMATCH

```
#define STATE_WINDING_RES_MISMATCH 0x0100000
```

The difference between winding resistances is too large.

This usually happens with a damaged stepper motor with partially short-circuited windings.

6.1.2.163 SYNCIN_ENABLED

```
#define SYNCIN_ENABLED 0x01
```

Synchronization in mode is enabled if this flag is set.

6.1.2.164 SYNCIN_GOTOPOSITION

```
#define SYNCIN_GOTOPOSITION 0x04
```

The engine is going to the position specified in Position and uPosition if this flag is set.

And it is shifting on the Position and uPosition if this flag is unset

6.1.2.165 SYNCIN_INVERT

```
#define SYNCIN_INVERT 0x02
```

Trigger on falling edge if flag is set, on rising edge otherwise.

6.1.2.166 SYNCOUT_ENABLED

```
#define SYNCOUT_ENABLED 0x01
```

The synchronization out pin follows the synchronization logic if the flag is set.

Otherwise, it is governed by the SYNCOUT_STATE flag.

6.1.2.167 SYNCOUT_IN_STEPS

```
#define SYNCOUT_IN_STEPS 0x08
```

Use motor steps or encoder pulses instead of milliseconds for output pulse generation if the flag is set.

6.1.2.168 SYNCOUT_INVERT

```
#define SYNCOUT_INVERT 0x04
```

The low level is active if the flag is set.

Otherwise, the high level is active.

6.1.2.169 SYNCOUT_ONPERIOD

```
#define SYNCOUT_ONPERIOD 0x40
```

Generate a synchronization pulse every SyncOutPeriod encoder pulses.

6.1.2.170 SYNCOUT_ONSTART

```
#define SYNCOUT_ONSTART 0x10
```

Generate a synchronization pulse when movement starts.

6.1.2.171 SYNCOUT_ONSTOP

```
#define SYNCOUT_ONSTOP 0x20
```

Generate a synchronization pulse when movement stops.

6.1.2.172 SYNCOUT_STATE

```
#define SYNCOUT_STATE 0x02
```

When the output state is fixed by the negative SYNCOUT_ENABLED flag, the pin state is in accordance with this flag state.

6.1.2.173 UART_PARITY_BITS

```
#define UART_PARITY_BITS 0x03
```

Bits of the parity.

6.1.2.174 WIND_A_STATE_ABSENT

```
#define WIND_A_STATE_ABSENT 0x00
```

Winding A is disconnected.

6.1.2.175 WIND_A_STATE_MALFUNC

```
#define WIND_A_STATE_MALFUNC 0x02
```

Winding A is short-circuited.

6.1.2.176 WIND_A_STATE_OK

```
#define WIND_A_STATE_OK 0x03
```

Winding A is connected and working properly.

6.1.2.177 WIND_A_STATE_UNKNOWN

```
#define WIND_A_STATE_UNKNOWN 0x01
```

Winding A state is unknown.

6.1.2.178 WIND_B_STATE_ABSENT

```
#define WIND_B_STATE_ABSENT 0x00
```

Winding B is disconnected.

6.1.2.179 WIND_B_STATE_MALFUNC

```
#define WIND_B_STATE_MALFUNC 0x20
```

Winding B is short-circuited.

6.1.2.180 WIND_B_STATE_OK

```
#define WIND_B_STATE_OK 0x30
```

Winding B is connected and working properly.

6.1.2.181 WIND_B_STATE_UNKNOWN

```
#define WIND_B_STATE_UNKNOWN 0x10
```

Winding B state is unknown.

6.1.2.182 XIMC_API

```
#define XIMC_API
```

Library import macro. Macros allows to automatically import function from shared library. It automatically expands to `__declspec(dllimport)` on `msvc` when including header file.

6.1.2.183 XIMC_CALLCONV

```
#define XIMC_CALLCONV
```

Library calling convention macros.

6.1.2.184 XIMC_RETTYPE

```
#define XIMC_RETTYPE void*
```

Thread return type.

6.1.3 Typedef Documentation

6.1.3.1 calibration_t

```
typedef struct calibration_t calibration_t
```

Calibration structure

Where to find all values for calculations?

- XILab (don't forget to load the profile for your positioner. The profile should match the full name of your positioner, e.g., 8MT173-25-MEn1.cfg):
 - In XILab settings, go to the "User units" tab. Divide the second number by the first one — this is your A coefficient.
 - In the "DC motor" / "BLDC motor" / "Stepper motor" tab (depending on your motor type), find the "Encoder counts per turn" field and use it in the formula for calculating coefficient B.
- Profile file (open with any text editor; make sure it matches the full name of your positioner, e.g., 8MT173-25-MEn1.cfg):
 - Find Step_multiplier= and Unit_multiplier=. Divide the second by the first — this is your A coefficient.
 - Find Encoder_CPT= and use this value in the B coefficient formula instead of ENCODER_↵ COUNTS_PER_TURN.

How to calculate Speed, Accel, Decel, and AntiplaySpeed in user units when using a stepper motor with encoder or DC/BLDC motor?

1. Load the correct profile in XILab for your positioner (e.g., 8MT173-25-MEn1.cfg).
 2. Enable Feedback encoder mode if not already enabled.
 3. Enter speed in the "Working speed" field in user units.
 4. In the "User units" tab, disable the "User units" flag to see RPM.
 5. Multiply the RPM value from "Working speed" by coefficient B. Example: $480 * 0.0000009375 = 0.00045$. This value for the 8MT173-25-MEn1 in Encoder mode equals 2 mm/s.
- Acceleration, deceleration, and antiplay speed are calculated the same way.

6.1.3.2 device_network_information_t

```
typedef struct device_network_information_t device_network_information_t
```

Device network information structure.

6.1.3.3 logging_callback_t

```
typedef void(XIMC_CALLCONV * logging_callback_t) (int loglevel, const wchar_t *message, void *user_data)
```

Logging callback prototype

Parameters

<i>loglevel</i>	a loglevel
<i>message</i>	a message

6.1.4 Function Documentation

6.1.4.1 close_device()

```
result_t XIMC_API close_device (  
    device_t * id)
```

Close specified device

Parameters

<i>id</i>	an identifier of device
-----------	-------------------------

Note

The id parameter in this function is a C pointer, unlike most library functions that use this parameter

6.1.4.2 command_clear_fram()

```
result_t XIMC_API command_clear_fram (  
    device_t id)
```

Clear controller FRAM.

Can be used by manufacturer only

Parameters

<i>id</i>	an identifier of device
-----------	-------------------------

6.1.4.3 command_eeread_settings()

```
result_t XIMC_API command_eeread_settings (  
    device_t id)
```

Read settings from the stage's EEPROM to the controller's RAM.

This operation is performed automatically at the connection of the stage with an EEPROM to the controller.
Can be used by the manufacturer only.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.4 `command_eesave_settings()`

```
result_t XIMC_API command_eesave_settings (  
    device_t id)
```

Save settings from the controller's RAM to the stage's EEPROM.

Can be used by the manufacturer only.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.5 `command_home()`

```
result_t XIMC_API command_home (  
    device_t id)
```

Moving to home position.

Moving algorithm:

- 1) Moves the motor according to the speed FastHome, uFastHome and flag HOME_DIR_FAST until the limit switch if the HOME_STOP_ENDS flag is set. Or moves the motor until the input synchronization signal occurs if the flag HOME_STOP_SYNC is set. Or moves until the revolution sensor signal occurs if the flag HOME_STOP_REV_SN is set.
- 2) Then moves according to the speed SlowHome, uSlowHome and flag HOME_DIR_SLOW until the input clock signal occurs if the flag HOME_MV_SEC is set. If the flag HOME_MV_SEC is reset, skip this step.
- 3) Then shifts the motor according to the speed FastHome, uFastHome and the flag HOME_DIR_SLOW by HomeDelta distance, uHomeDelta.

See GHOM/SHOM commands' description for details on home flags.

Moving settings can be set by set_home_settings/set_home_settings_calb.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

See also

[home_settings_t](#)
[get_home_settings](#)
[set_home_settings](#)

6.1.4.6 `command_homezero()`

```
result_t XIMC_API command_homezero (  
    device_t id)
```

Make home command, wait until it is finished and make zero command.

This is a convinient way to calibrate zero position.

Parameters

	<i>id</i>	an identifier of device
out	<i>ret</i>	RESULT_OK if controller has finished home & zero correctly or result of first controller query that returned anything other than RESULT_OK.

6.1.4.7 command_left()

```
result_t XIMC_API command_left (
    device_t id)
```

Start continuous moving to the left.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.8 command_loft()

```
result_t XIMC_API command_loft (
    device_t id)
```

Upon receiving the command "loft", the engine is shifted from the current position to a distance Antiplay defined in engine settings.

Then moves to the initial position.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.9 command_move()

```
result_t XIMC_API command_move (
    device_t id,
    int Position,
    int uPosition)
```

Move to position.

Upon receiving the command "move" the engine starts to move with pre-set parameters (speed, acceleration, retention), to the point specified by Position and uPosition. uPosition sets the microstep position of a stepper motor. In the case of DC motor, this field is ignored.

Parameters

<i>id</i>	An identifier of a device
<i>Position</i>	position to move.
<i>uPosition</i>	the fractional part of the position to move, in microsteps. The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings).

6.1.4.10 command_move_calb()

```
result_t XIMC_API command_move_calb (
    device_t id,
    float Position,
    const calibration_t * calibration)
```

Move to position using user units.

Upon receiving the command "move" the engine starts to move with preset parameters (speed, acceleration, retention), to the point specified by Position.

Parameters

<i>id</i>	An identifier of a device
<i>Position</i>	position to move.
<i>calibration</i>	user unit settings

Note

The parameter Position is adjusted by the correction table.

6.1.4.11 command_movr()

```
result_t XIMC_API command_movr (
    device_t id,
    int DeltaPosition,
    int uDeltaPosition)
```

Shift by a set offset.

Upon receiving the command "movr", the engine starts to move with preset parameters (speed, acceleration, hold) left or right (depending on the sign of DeltaPosition). It moves by the number of steps specified in the fields DeltaPosition and uDeltaPosition. uDeltaPosition sets the microstep offset for a stepper motor. In the case of a DC motor, this field is ignored.

Parameters

<i>DeltaPosition</i>	shift from initial position.
<i>uDeltaPosition</i>	the fractional part of the offset shift, in microsteps. The microstep size and the range of valid values for this field depend on the selected step division mode (see the MicrostepMode field in engine_settings).
<i>id</i>	An identifier of a device

6.1.4.12 command_movr_calb()

```
result_t XIMC_API command_movr_calb (
    device_t id,
    float DeltaPosition,
    const calibration_t * calibration)
```

Shift by a set offset using user units.

Upon receiving the command "movr", the engine starts to move with preset parameters (speed, acceleration, hold) left or right (depending on the sign of DeltaPosition). It moves by the distance specified in the field DeltaPosition.

Parameters

<i>DeltaPosition</i>	shift from initial position.
<i>id</i>	An identifier of a device
<i>user</i>	unit calibration settings

Note

The final coordinate is calculated using *DeltaPosition* and adjusted by the correction table. However, the correction cannot be done if the motor moves. *movr* sets the target position equal to the current target position, shifted by delta. But the library can't determine the current target position while moving. So there is no possibility of calculating the final position and correcting it with the correction table.

6.1.4.13 command_power_off()

```
result_t XIMC_API command_power_off (
    device_t id)
```

Immediately power off the motor regardless its state.

Shouldn't be used during motion as the motor could be powered on again automatically to continue movement. The command is designed to manually power off the motor. When automatic power off after stop is required, use the power management system.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

See also

[get_power_settings](#)

[set_power_settings](#)

6.1.4.14 command_read_robust_settings()

```
result_t XIMC_API command_read_robust_settings (
    device_t id)
```

Read important settings (calibration coefficients, etc.) from the controller's flash memory to the controller's RAM, replacing previous data in the RAM.

Manufacturer only.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.15 command_read_settings()

```
result_t XIMC_API command_read_settings (
    device_t id)
```

Read all settings from the controller's flash memory to the controller's RAM, replacing previous data in the RAM.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.16 `command_reset()`

```
result_t XIMC_API command_reset (  
    device_t id)
```

Reset controller.

Can be used by manufacturer only

Parameters

<i>id</i>	an identifier of device
-----------	-------------------------

6.1.4.17 `command_right()`

```
result_t XIMC_API command_right (  
    device_t id)
```

Start continuous moving to the right.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.18 `command_save_robust_settings()`

```
result_t XIMC_API command_save_robust_settings (  
    device_t id)
```

Save important settings (calibration coefficients, etc.) from the controller's RAM to the controller's flash memory, replacing previous data in the flash memory.

Manufacturer only.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.19 `command_save_settings()`

```
result_t XIMC_API command_save_settings (  
    device_t id)
```

Save all settings from the controller's RAM to the controller's flash memory, replacing previous data in the flash memory.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.20 `command_sstp()`

```
result_t XIMC_API command_sstp (  
    device_t id)
```

Soft stop the engine.

The motor is slowing down with the deceleration specified in `move_settings`.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.21 `command_start_measurements()`

```
result_t XIMC_API command_start_measurements (  
    device_t id)
```

Start measurements and buffering of speed and the speed error (target speed minus real speed).

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.22 `command_stop()`

```
result_t XIMC_API command_stop (  
    device_t id)
```

Immediately stops the engine, moves it to the STOP state, and sets switches to BREAK mode (windings are short-circuited).

The holding regime is deactivated for DC motors, keeping current in the windings for stepper motors (to control it, see Power management settings).

When this command is called, the ALARM flag is reset.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.23 `command_update_firmware()`

```
result_t XIMC_API command_update_firmware (  
    const char * uri,  
    const uint8_t * data,  
    uint32_t data_size)
```

Update firmware.

Manufacturer only. Service command

Parameters

<i>uri</i>	a uri of device
<i>data</i>	firmware byte stream
<i>data_size</i>	size of byte stream

6.1.4.24 `command_wait_for_stop()`

```
result_t XIMC_API command_wait_for_stop (
    device_t id,
    uint32_t refresh_interval_ms)
```

Wait for stop

Parameters

	<i>id</i>	an identifier of device
	<i>refresh_interval_ms</i>	Status refresh interval. The function waits this number of milliseconds between get_status requests to the controller. Recommended value of this parameter is 10 ms. Use values of less than 3 ms only when necessary - small refresh interval values do not significantly increase response time of the function, but they create substantially more traffic in controller-computer data channel.
out	<i>ret</i>	RESULT_OK if controller has stopped and result of the first get_status command which returned anything other than RESULT_OK otherwise.

6.1.4.25 `command_zero()`

```
result_t XIMC_API command_zero (
    device_t id)
```

Sets the current position to 0.

Sets the target position of the move command and the movr command to zero for all cases except for movement to the target position. In the latter case, the target position is calculated so that the absolute position of the destination stays the same. For example, if we were at 400 and moved to 500, then the command Zero makes the current position 0 and the position of the destination 100. It does not change the mode of movement. If the motion is carried, it continues, and if the engine is in the "hold", the type of retention remains.

Parameters

<i>id</i>	An identifier of a device
-----------	---------------------------

6.1.4.26 enumerate_devices()

```
device_enumeration_t XIMC_API enumerate_devices (
    int enumerate_flags,
    const char * hints)
```

Search and list of available devices.

By default, it creates a list of devices connected to this computer and presented as COM ports. Additionally, you can enable the search for network devices. Devices found in the local network will be included in the same list.

To obtain information from the collected list, use the corresponding functions with the `get_enumerate_` prefix and the received `device_enumeration` identifier.

After finishing working with the list of found devices, you should free up memory using the `free_enumerate_devices()` function.

Parameters

in	<i>enumerate_flags</i>	<p>a set of flags that specify search modes. Flags can be used together via bitwise "OR":</p> <ul style="list-style-type: none">• <code>ENUMERATE_NETWORK</code> - enables searching for network devices. If the flag is set, network devices will be added to the general list. If the flag is not set, the list will only include devices connected to this computer.• <code>ENUMERATE_ALL_COM</code> - when enabled, queries all COM port devices in the system. When disabled, queries only devices whose names match the XIMC device mask ("XIMC Motor Controller" in Windows, <code>/dev/ximc/</code> and <code>/dev/ttyACM/</code> on Linux/Mac).• <code>ENUMERATE_PROBE</code> - enables checking of devices and collecting additional information (serial number, version, model, name...). If this flag is set, only devices that are guaranteed to be open will be added to the list, but devices connected via RS232 converters may not be included. If the flag is not set, the list will include more devices (in particular, devices explicitly listed in <code>hints</code> will be included), but availability and compatibility with this library is not guaranteed.
----	------------------------	--

Parameters

in	<i>hints</i>	<p>additional information to improve the search efficiency. It makes sense to use in case of complex network configuration, when automatic search may not find everything. Format - string "key1=value1\nkey2=value2". Unknown keys are ignored. One key can have several values, which are listed separated by commas: key=value1,value2,value3. Valid keys:</p> <ul style="list-style-type: none"> • <code>addr</code> - list of URLs of network controllers or servers with connected controllers. The field is used together with the <code>ENUMERATE_NETWORK</code> flag. Protocols and address formats: <ul style="list-style-type: none"> - <code>xi-tcp://<ip-address></code> - network controllers and controllers connected via Ethernet-RS232 converters. If the <code>ENUMERATE_PROBE</code> flag is not set, all listed devices will be included in the list. - <code>xi-net://<ip-address></code> - network multi-axis systems, xi-net servers. The request for information about the availability of controllers at these addresses will be made regardless of the results of the automatic network search procedure. • <code>adapter_addr</code> - list of IP addresses of local network adapters through which the search should be performed. If the key is missing or no adapters are specified, the search is performed on all adapters.
----	--------------	---

Returns

`device_enumeration` - device list identifier. Used to obtain information about devices using functions with `get_enumerate_` prefixes.

Examples of use:

```
// Search for local devices without checking
device_enumeration_t device_enumeration = enumerate_devices(0, "");
// Search for local devices with verification
device_enumeration_t device_enumeration = enumerate_devices(ENUMERATE_PROBE, "");
// Fully automatic search for local and network devices
device_enumeration_t device_enumeration = enumerate_devices(ENUMERATE_NETWORK, "");
// Search for local and network devices
// using the local computer adapter with the address 192.168.0.100
// and explicit requests to the network controller with the address 192.168.0.11
// and the xi-net server with the address 192.168.0.10
device_enumeration_t device_enumeration = enumerate_devices(
    ENUMERATE_NETWORK,
    "addr=192.168.0.10,xi-tcp://192.168.0.11\nadapter_addr=192.168.0.100"
);
```

6.1.4.27 free_enumerate_devices()

```
result_t XIMC_API free_enumerate_devices (
    device_enumeration_t device_enumeration)
```

Free memory returned by `enumerate_devices`.

Parameters

in	<i>device_enumeration</i>	opaque pointer to an enumeration device data
----	---------------------------	--

6.1.4.28 get_accessories_settings()

```
result_t XIMC_API get_accessories_settings (
    device_t id,
    accessories_settings_t * accessories_settings)
```

Deprecated.

Read additional accessory information from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>accessories_settings</i>	structure contains information about additional accessories

6.1.4.29 get_analog_data()

```
result_t XIMC_API get_analog_data (
    device_t id,
    analog_data_t * analog_data)
```

Read the analog data structure that contains raw analog data from the embedded ADC.

This function is used for device testing and deep recalibration by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
out	<i>analog_data</i>	analog data coefficients

6.1.4.30 get_bootloader_version()

```
result_t XIMC_API get_bootloader_version (
    device_t id,
    unsigned int * Major,
    unsigned int * Minor,
    unsigned int * Release)
```

Read the controller's bootloader version.

Parameters

	<i>id</i>	An identifier of a device
out	<i>Major</i>	major version
out	<i>Minor</i>	minor version
out	<i>Release</i>	release version

6.1.4.31 get_brake_settings()

```
result_t XIMC_API get_brake_settings (
    device_t id,
    brake_settings_t * brake_settings)
```

Read break control settings.

Parameters

	<i>id</i>	An identifier of a device
out	<i>brake_settings</i>	structure contains settings of brake control

6.1.4.32 get_calibration_settings()

```
result_t XIMC_API get_calibration_settings (
    device_t id,
    calibration_settings_t * calibration_settings)
```

Read calibration settings.

Manufacturer only. This function reads the structure with calibration settings. These settings are used to convert bare ADC values to winding currents in mA and the full current in mA. Parameters are grouped into pairs, XXX_A and XXX_B, representing linear equation coefficients. The first one is the slope, the second one is the constant term. Thus, $\text{XXX_Current[mA]} = \text{XXX_A[mA/ADC]} * \text{XXX_ADC_CODE[ADC]} + \text{XXX_B[mA]}$.

See also

[calibration_settings_t](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>calibration_settings</i>	calibration settings

6.1.4.33 get_chart_data()

```
result_t XIMC_API get_chart_data (
    device_t id,
    chart_data_t * chart_data)
```

Return device electrical parameters, useful for charts.

A useful function that fills the structure with a snapshot of the controller voltages and currents.

See also

[chart_data_t](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>chart_data</i>	structure with a snapshot of controller parameters.

6.1.4.34 get_control_settings()

```
result_t XIMC_API get_control_settings (
    device_t id,
    control_settings_t * control_settings)
```

Read motor control settings.

In case of CTL_MODE=1, joystick motor control is enabled. In this mode, while the joystick is maximally displaced, the engine tends to move at MaxSpeed[i]. i=0 if another value hasn't been set at the previous usage. To change the speed index "i", use the buttons.

In case of CTL_MODE=2, the motor is controlled by the left/right buttons. When you click on the button, the motor starts moving in the appropriate direction at a speed MaxSpeed[0]. After Timeout[i], motor moves at speed MaxSpeed[i+1]. At the transition between MaxSpeed[i] and MaxSpeed[i+1] the motor just accelerates/decelerates as usual.

Parameters

	<i>id</i>	An identifier of a device
out	<i>control_settings</i>	structure contains settings motor control by joystick or buttons left/right.

6.1.4.35 get_control_settings_calb()

```
result_t XIMC_API get_control_settings_calb (
    device_t id,
    control_settings_calb_t * control_settings_calb,
    const calibration_t * calibration)
```

Read motor control settings with user units.

In case of CTL_MODE=1, the joystick motor control is enabled. In this mode, while the joystick is maximally displaced, the engine tends to move at MaxSpeed[i]. i=0 if another value hasn't been set at the previous usage. To change the speed index "i", use the buttons.

In case of CTL_MODE=2, the motor is controlled by the left/right buttons. When you click on the button, the motor starts moving in the appropriate direction at a speed MaxSpeed[0]. After Timeout[i], motor moves at speed MaxSpeed[i+1]. At the transition between MaxSpeed[i] and MaxSpeed[i+1] the motor just accelerates/decelerates as usual.

Parameters

	<i>id</i>	An identifier of a device
out	<i>control_settings_calb</i>	structure contains user unit motor control settings.
	<i>calibration</i>	user unit settings

6.1.4.36 get_controller_name()

```
result_t XIMC_API get_controller_name (
    device_t id,
    controller_name_t * controller_name)
```

Read user's controller name and internal settings from the FRAM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>controller_name</i>	structure contains previously set user controller name

6.1.4.37 get_ctp_settings()

```
result_t XIMC_API get_ctp_settings (
    device_t id,
    ctp_settings_t * ctp_settings)
```

Read control position settings (used with stepper motor only).

When controlling the step motor with an encoder (CTP_BASE=0), it is possible to detect the loss of steps. The controller knows the number of steps per revolution (GENG::StepsPerRev) and the encoder resolution (GFBS::IPT). When the control is enabled (CTP_ENABLED is set), the controller stores the current position in the steps of SM and the current position of the encoder. Next, the encoder position is converted into steps at each step, and if the difference between the current position in steps and the encoder position is greater than CTPMinError, the flag STATE_CTP_ERROR is set.

Alternatively, the stepper motor may be controlled with the speed sensor (CTP_BASE 1). In this mode, at the active edges of the input clock, the controller stores the current value of steps. Then, at each revolution, the controller checks how many steps have been passed. When the difference is over the CTPMinError, the STATE_CTP_ERROR flag is set.

Parameters

	<i>id</i>	An identifier of a device
out	<i>ctp_settings</i>	structure contains position control settings.

6.1.4.38 get_debug_read()

```
result_t XIMC_API get_debug_read (
    device_t id,
    debug_read_t * debug_read)
```

Read data from firmware for debug purpose.

Manufacturer only. Its use depends on context, firmware version and previous history.

Parameters

	<i>id</i>	An identifier of a device
out	<i>debug_read</i>	Debug data.

6.1.4.39 get_device_count()

```
int XIMC_API get_device_count (
    device_enumeration_t device_enumeration)
```

Get device count.

Parameters

in	<i>device_enumeration</i>	opaque pointer to an enumeration device data
----	---------------------------	--

6.1.4.40 get_device_information()

```
result_t XIMC_API get_device_information (
    device_t id,
    device_information_t * device_information)
```

Return device information.

All fields must point to allocated string buffers with at least 10 bytes. Works with both raw or initialized device.

Parameters

	<i>id</i>	an identifier of device
out	<i>device_information</i>	device information Device information.

See also

[get_device_information](#)

6.1.4.41 get_device_name()

```
pchar XIMC_API get_device_name (
    device_enumeration_t device_enumeration,
    int device_index)
```

Get device name from the device enumeration.

Returns *device_index* device name.

Parameters

in	<i>device_enumeration</i>	opaque pointer to an enumeration device data
in	<i>device_index</i>	device index

6.1.4.42 `get_edges_settings()`

```
result_t XIMC_API get_edges_settings (
    device_t id,
    edges_settings_t * edges_settings)
```

Read border and limit switches settings.

See also

[set_edges_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>edges_settings</i>	edges settings, types of borders, motor behavior and electrical behavior of limit switches

6.1.4.43 `get_edges_settings_calb()`

```
result_t XIMC_API get_edges_settings_calb (
    device_t id,
    edges_settings_calb_t * edges_settings_calb,
    const calibration_t * calibration)
```

Read border and limit switches settings in user units.

See also

[set_edges_settings_calb](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>edges_settings_calb</i>	edges settings, types of borders, motor behavior and electrical behavior of limit switches
	<i>calibration</i>	user unit settings

Note

Attention! Some parameters of the `edges_settings_calb` structure are corrected by the coordinate correction table.

6.1.4.44 get_emf_settings()

```
result_t XIMC_API get_emf_settings (  
    device_t id,  
    emf_settings_t * emf_settings)
```

Read electromechanical settings.

The settings are different for different stepper motors.

See also

[set_emf_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>emf_settings</i>	EMF settings

6.1.4.45 get_encoder_information()

```
result_t XIMC_API get_encoder_information (  
    device_t id,  
    encoder_information_t * encoder_information)
```

Deprecated.

Read encoder information from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>encoder_information</i>	structure contains information about encoder

6.1.4.46 get_encoder_settings()

```
result_t XIMC_API get_encoder_settings (  
    device_t id,  
    encoder_settings_t * encoder_settings)
```

Deprecated.

Read encoder settings from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>encoder_settings</i>	structure contains encoder settings

6.1.4.47 `get_engine_advanced_setup()`

```
result_t XIMC_API get_engine_advanced_setup (
    device_t id,
    engine_advanced_setup_t * engine_advanced_setup)
```

Read engine advanced settings.

Manufacturer only.

See also

[set_engine_advanced_setup](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>engine_advanced_setup</i>	EAS settings

6.1.4.48 `get_engine_settings()`

```
result_t XIMC_API get_engine_settings (
    device_t id,
    engine_settings_t * engine_settings)
```

Read engine settings.

This function reads the structure containing a set of useful motor settings stored in the controller's memory. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics.

See also

[set_engine_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>engine_settings</i>	engine settings

6.1.4.49 `get_engine_settings_calb()`

```
result_t XIMC_API get_engine_settings_calb (
    device_t id,
    engine_settings_calb_t * engine_settings_calb,
    const calibration_t * calibration)
```

Read user unit engine settings.

This function reads the structure containing a set of useful motor settings stored in the controller's memory. These settings specify the motor shaft movement algorithm, list of limitations and rated characteristics.

See also

[set_engine_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>engine_settings_calb</i>	engine settings
	<i>calibration</i>	user unit settings

6.1.4.50 get_entype_settings()

```
result_t XIMC_API get_entype_settings (
    device_t id,
    entype_settings_t * entype_settings)
```

Return engine type and driver type.

Parameters

	<i>id</i>	An identifier of a device
out	<i>entype_settings</i>	structure contains motor type and power driver type settings

6.1.4.51 get_enumerate_device_controller_name()

```
result_t XIMC_API get_enumerate_device_controller_name (
    device_enumeration_t device_enumeration,
    int device_index,
    controller_name_t * controller_name)
```

Get controller name from the device enumeration.

Returns *device_index* device controller name.

Parameters

in	<i>device_enumeration</i>	opaque pointer to an enumeration device data
in	<i>device_index</i>	device index
out	<i>controller_name</i>	controller name

6.1.4.52 get_enumerate_device_information()

```
result_t XIMC_API get_enumerate_device_information (
    device_enumeration_t device_enumeration,
    int device_index,
    device_information_t * device_information)
```

Get device information from the device enumeration.

Returns *device_index* device information.

Parameters

in	<i>device_enumeration</i>	opaque pointer to an enumeration device data
in	<i>device_index</i>	device index
out	<i>device_information</i>	device information data

6.1.4.53 get_enumerate_device_network_information()

```
result_t XIMC_API get_enumerate_device_network_information (
    device_enumeration_t device_enumeration,
    int device_index,
    device_network_information_t * device_network_information)
```

Get device network information from the device enumeration.

Returns *device_index* device network information.

Parameters

in	<i>device_enumeration</i>	opaque pointer to an enumeration device data
in	<i>device_index</i>	device index
out	<i>device_network_information</i>	device network information data

6.1.4.54 get_enumerate_device_serial()

```
result_t XIMC_API get_enumerate_device_serial (
    device_enumeration_t device_enumeration,
    int device_index,
    uint32_t * serial)
```

Get device serial number from the device enumeration.

Returns *device_index* device serial number.

Parameters

in	<i>device_enumeration</i>	opaque pointer to an enumeration device data
in	<i>device_index</i>	device index
out	<i>serial</i>	device serial number

6.1.4.55 get_enumerate_device_stage_name()

```
result_t XIMC_API get_enumerate_device_stage_name (
    device_enumeration_t device_enumeration,
    int device_index,
    stage_name_t * stage_name)
```

Get stage name from the device enumeration.

Returns *device_index* device stage name.

Parameters

in	<i>device_enumeration</i>	opaque pointer to an enumeration device data
in	<i>device_index</i>	device index
out	<i>stage_name</i>	stage name

6.1.4.56 `get_extended_settings()`

```
result_t XIMC_API get_extended_settings (  
    device_t id,  
    extended_settings_t * extended_settings)
```

Read extended settings.

Currently, it is not in use.

See also

[set_extended_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>extended_settings</i>	EST settings

6.1.4.57 `get_extio_settings()`

```
result_t XIMC_API get_extio_settings (  
    device_t id,  
    extio_settings_t * extio_settings)
```

Read EXTIO settings.

This function reads a structure with a set of EXTIO settings from the controller's memory.

See also

[set_extio_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>extio_settings</i>	EXTIO settings

6.1.4.58 `get_feedback_settings()`

```
result_t XIMC_API get_feedback_settings (  
    device_t id,  
    feedback_settings_t * feedback_settings)
```

Feedback settings.

Parameters

	<i>id</i>	An identifier of a device
out	<i>IPS</i>	number of encoder counts per shaft revolution. Range: 1..65535. The field is obsolete, it is recommended to write 0 to IPS and use the extended CountsPerTurn field. You may need to update the controller firmware to the latest version.
out	<i>FeedbackType</i>	type of feedback
out	<i>FeedbackFlags</i>	flags of feedback
out	<i>CountsPerTurn</i>	number of encoder counts per shaft revolution. Range: 1..4294967295. To use the CountsPerTurn field, write 0 in the IPS field, otherwise the value from the IPS field will be used.

6.1.4.59 get_firmware_version()

```
result_t XIMC_API get_firmware_version (
    device_t id,
    unsigned int * Major,
    unsigned int * Minor,
    unsigned int * Release)
```

Read the controller's firmware version.

Parameters

	<i>id</i>	An identifier of a device
out	<i>Major</i>	major version
out	<i>Minor</i>	minor version
out	<i>Release</i>	release version

6.1.4.60 get_gear_information()

```
result_t XIMC_API get_gear_information (
    device_t id,
    gear_information_t * gear_information)
```

Deprecated.

Read gear information from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>gear_information</i>	structure contains information about step gearhead

6.1.4.61 get_gear_settings()

```
result_t XIMC_API get_gear_settings (
    device_t id,
    gear_settings_t * gear_settings)
```

Deprecated.

Read gear settings from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>gear_settings</i>	structure contains step gearhead settings

6.1.4.62 get_globally_unique_identifier()

```
result_t XIMC_API get_globally_unique_identifier (  
    device_t id,  
    globally_unique_identifier_t * globally_unique_identifier)
```

This value is unique to each individual device, but is not a random value.

Manufacturer only. This unique device identifier can be used to initiate secure boot processes or as a serial number for USB or other end applications.

Parameters

	<i>id</i>	An identifier of a device
out	<i>globally_unique_identifier</i>	the result of fields 0-3 concatenated defines the unique 128-bit device identifier.

6.1.4.63 get_hallsensor_information()

```
result_t XIMC_API get_hallsensor_information (  
    device_t id,  
    hallsensor_information_t * hallsensor_information)
```

Deprecated.

Read hall sensor information from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>hallsensor_information</i>	structure contains information about hall sensor

6.1.4.64 get_hallsensor_settings()

```
result_t XIMC_API get_hallsensor_settings (  
    device_t id,  
    hallsensor_settings_t * hallsensor_settings)
```

Deprecated.

Read hall sensor settings from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>hallsensor_settings</i>	structure contains hall sensor settings

6.1.4.65 get_home_settings()

```
result_t XIMC_API get_home_settings (
    device_t id,
    home_settings_t * home_settings)
```

Read home settings.

This function reads the structure with home position settings.

See also

[home_settings_t](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>home_settings</i>	calibrating position settings

6.1.4.66 get_home_settings_calb()

```
result_t XIMC_API get_home_settings_calb (
    device_t id,
    home_settings_calb_t * home_settings_calb,
    const calibration_t * calibration)
```

Read user unit home settings.

This function reads the structure with home position settings.

See also

[home_settings_calb_t](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>home_settings_calb</i>	calibrating position settings
	<i>calibration</i>	user unit settings

6.1.4.67 get_init_random()

```
result_t XIMC_API get_init_random (
    device_t id,
    init_random_t * init_random)
```

Read a random number from the controller.

Manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
out	<i>init_random</i>	random sequence generated by the controller

6.1.4.68 get_joystick_settings()

```
result_t XIMC_API get_joystick_settings (
    device_t id,
    joystick_settings_t * joystick_settings)
```

Read joystick settings.

If joystick position falls outside DeadZone limits, a movement begins. The speed is defined by the joystick's position in the range from the DeadZone limit to the maximum deviation. Joystick positions inside DeadZone limits correspond to zero speed (a "soft stop" command is issued continuously), and positions beyond Low and High limits correspond to MaxSpeed[i] or -MaxSpeed[i] (see command SCTL), where $i = 0$ by default and can be changed with the left/right buttons (see command SCTL). If the next speed in the list is zero (both integer and microstep parts), the button press is ignored. The first speed in the list shouldn't be zero. DeadZone is defined in 0.1% units. See the Joystick control section on https://doc.xisupport.com/en/8smc5-usb/8SMCn-USB/Technical_specification/Additional_features/Joystick_control.html for more information.

Parameters

	<i>id</i>	An identifier of a device
out	<i>joystick_settings</i>	structure contains joystick settings

6.1.4.69 get_measurements()

```
result_t XIMC_API get_measurements (
    device_t id,
    measurements_t * measurements)
```

A command to read the data buffer to build a speed graph and a speed error graph.

Filling the buffer starts with the command "start_measurements". The buffer holds 25 points; the points are taken with a period of 1 ms. To create a robust system, read data every 20 ms. If the buffer is full, it is recommended to repeat the readings every 5 ms until the buffer again becomes filled with 20 points.

To stop measurements just stop reading data. After buffer overflow measurements will stop automatically.

See also

[measurements_t](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>measurements</i>	structure with buffer and its length.

6.1.4.70 get_motor_information()

```
result_t XIMC_API get_motor_information (
    device_t id,
    motor_information_t * motor_information)
```

Deprecated.

Read motor information from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>motor_information</i>	structure contains motor information

6.1.4.71 get_motor_settings()

```
result_t XIMC_API get_motor_settings (
    device_t id,
    motor_settings_t * motor_settings)
```

Deprecated.

Read motor settings from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>motor_settings</i>	structure contains motor settings

6.1.4.72 get_move_settings()

```
result_t XIMC_API get_move_settings (
    device_t id,
    move_settings_t * move_settings)
```

Movement settings read command (speed, acceleration, threshold, etc.).

Parameters

	<i>id</i>	An identifier of a device
out	<i>move_settings</i>	structure contains move settings: speed, acceleration, deceleration etc.

6.1.4.73 get_move_settings_calb()

```
result_t XIMC_API get_move_settings_calb (
    device_t id,
    move_settings_calb_t * move_settings_calb,
    const calibration_t * calibration)
```

User unit movement settings read command (speed, acceleration, threshold, etc.).

Parameters

	<i>id</i>	An identifier of a device
out	<i>move_settings_calb</i>	structure contains move settings: speed, acceleration, deceleration etc.
	<i>calibration</i>	user unit settings

6.1.4.74 get_network_settings()

```
result_t XIMC_API get_network_settings (
    device_t id,
    network_settings_t * network_settings)
```

Read network settings.

Manufacturer only. This function returns the current network settings.

See also

net_settings_t

Parameters

<i>DHCPEnabled</i>	DHCP enabled (1) or not (0)
<i>IPv4Address[4]</i>	Array[4] with an IP address
<i>SubnetMask[4]</i>	Array[4] with a subnet mask address
<i>DefaultGateway[4]</i>	Array[4] with a default gateway address

6.1.4.75 get_nonvolatile_memory()

```
result_t XIMC_API get_nonvolatile_memory (
    device_t id,
    nonvolatile_memory_t * nonvolatile_memory)
```

Read user data from FRAM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>nonvolatile_memory</i>	structure contains previously set user data.

6.1.4.76 get_password_settings()

```
result_t XIMC_API get_password_settings (
    device_t id,
    password_settings_t * password_settings)
```

Read the password.

Manufacturer only. This function reads the user password for the device's web-page.

See also

pwd_settings_t

Parameters

<i>UserPassword[20]</i>	Password for web-page
-------------------------	-----------------------

6.1.4.77 get_pid_settings()

```
result_t XIMC_API get_pid_settings (
    device_t id,
    pid_settings_t * pid_settings)
```

Read PID settings.

This function reads the structure containing a set of motor PID settings stored in the controller's memory. These settings specify the behavior of the PID routine for the positioner. These factors are slightly different for different positioners. All boards are supplied with the standard set of PID settings in the controller's flash memory.

See also

[set_pid_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>pid_settings</i>	PID settings

6.1.4.78 get_position()

```
result_t XIMC_API get_position (
    device_t id,
    get_position_t * the_get_position)
```

Reads the value position in steps and microsteps for stepper motor and encoder steps for all engines.

Parameters

	<i>id</i>	An identifier of a device
out	<i>the_get_position</i>	structure contains motor position.

6.1.4.79 get_position_calb()

```
result_t XIMC_API get_position_calb (
    device_t id,
    get_position_calb_t * the_get_position_calb,
    const calibration_t * calibration)
```

Reads position value in user units for stepper motor and encoder steps for all engines.

Parameters

	<i>id</i>	An identifier of a device
out	<i>the_get_position_calb</i>	structure contains motor position.
	<i>calibration</i>	user unit settings

Note

Attention! Some parameters of the `get_position_calb` structure are corrected by the coordinate correction table.

6.1.4.80 get_power_settings()

```
result_t XIMC_API get_power_settings (
    device_t id,
    power_settings_t * power_settings)
```

Read settings of step motor power control.

Used with a stepper motor only.

Parameters

	<i>id</i>	An identifier of a device
out	<i>power_settings</i>	structure contains settings of step motor power control

6.1.4.81 get_secure_settings()

```
result_t XIMC_API get_secure_settings (
    device_t id,
    secure_settings_t * secure_settings)
```

Read protection settings.

Parameters

	<i>id</i>	An identifier of a device
out	<i>secure_settings</i>	critical parameter settings to protect the hardware

See also

`status_t::flags`

6.1.4.82 get_serial_number()

```
result_t XIMC_API get_serial_number (
    device_t id,
    unsigned int * SerialNumber)
```

Read device serial number.

Parameters

	<i>id</i>	An identifier of a device
out	<i>SerialNumber</i>	serial number

6.1.4.83 get_stage_information()

```
result_t XIMC_API get_stage_information (  
    device_t id,  
    stage_information_t * stage_information)
```

Deprecated.

Read stage information from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>stage_information</i>	structure contains stage information

6.1.4.84 get_stage_name()

```
result_t XIMC_API get_stage_name (  
    device_t id,  
    stage_name_t * stage_name)
```

Read the user's stage name from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>stage_name</i>	structure contains the previously set user's stage name.

6.1.4.85 get_stage_settings()

```
result_t XIMC_API get_stage_settings (  
    device_t id,  
    stage_settings_t * stage_settings)
```

Deprecated.

Read stage settings from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>stage_settings</i>	structure contains stage settings

6.1.4.86 get_status()

```
result_t XIMC_API get_status (  
    device_t id,  
    status_t * status)
```

Return device state.

Parameters

	<i>id</i>	an identifier of device
out	<i>status</i>	structure with snapshot of controller status Device state. Useful structure that contains current controller status, including speed, position and boolean flags.

See also

[get_status](#)

6.1.4.87 get_status_calb()

```
result_t XIMC_API get_status_calb (  
    device_t id,  
    status_calb_t * status,  
    const calibration_t * calibration)
```

Return device state.

Parameters

	<i>id</i>	an identifier of device
out	<i>status</i>	structure with snapshot of controller status
	<i>calibration</i>	user unit settings Calibrated device state. Useful structure that contains current controller status, including speed, position and boolean flags.

See also

[get_status](#)

6.1.4.88 get_sync_in_settings()

```
result_t XIMC_API get_sync_in_settings (  
    device_t id,  
    sync_in_settings_t * sync_in_settings)
```

Read input synchronization settings.

This function reads the structure with a set of input synchronization settings, modes, periods and flags that specify the behavior of input synchronization. All boards are supplied with the standard set of these settings.

See also

[set_sync_in_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>sync_in_settings</i>	synchronization settings

6.1.4.89 get_sync_in_settings_calb()

```
result_t XIMC_API get_sync_in_settings_calb (
    device_t id,
    sync_in_settings_calb_t * sync_in_settings_calb,
    const calibration_t * calibration)
```

Read input user unit synchronization settings.

This function reads the structure with a set of input synchronization settings, modes, periods and flags that specify the behavior of input synchronization. All boards are supplied with the standard set of these settings.

See also

[set_sync_in_settings_calb](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>sync_in_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

6.1.4.90 get_sync_out_settings()

```
result_t XIMC_API get_sync_out_settings (
    device_t id,
    sync_out_settings_t * sync_out_settings)
```

Read output synchronization settings.

This function reads the structure containing a set of output synchronization settings, modes, periods and flags that specify the behavior of output synchronization. All boards are supplied with the standard set of these settings.

See also

[set_sync_out_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>sync_out_settings</i>	synchronization settings

6.1.4.91 get_sync_out_settings_calb()

```
result_t XIMC_API get_sync_out_settings_calb (
    device_t id,
    sync_out_settings_calb_t * sync_out_settings_calb,
    const calibration_t * calibration)
```

Read output user unit synchronization settings.

This function reads the structure containing a set of output synchronization settings, modes, periods and flags that specify the behavior of output synchronization. All boards are supplied with the standard set of these settings.

See also

[set_sync_in_settings_calb](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>sync_out_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

6.1.4.92 get_uart_settings()

```
result_t XIMC_API get_uart_settings (
    device_t id,
    uart_settings_t * uart_settings)
```

Read UART settings.

This function reads the structure containing UART settings.

See also

[uart_settings_t](#)

Parameters

	<i>Speed</i>	UART speed
out	<i>uart_settings</i>	UART settings

6.1.4.93 goto_firmware()

```
result_t XIMC_API goto_firmware (
    device_t id,
    uint8_t * ret)
```

Reboot to firmware

Parameters

	<i>id</i>	an identifier of device
out	<i>ret</i>	RESULT_OK, if reboot to firmware is possible. Reboot is done after reply to this command. RESULT_NO_FIRMWARE, if firmware is not found. RESULT_ALREADY_IN_FIRMWARE, if this command was sent when controller is already in firmware.

6.1.4.94 has_firmware()

```
result_t XIMC_API has_firmware (
    const char * uri,
    uint8_t * ret)
```

Check for firmware on device

Parameters

	<i>uri</i>	a uri of device
out	<i>ret</i>	non-zero if firmware existed

6.1.4.95 logging_callback_stderr_narrow()

```
void XIMC_API logging_callback_stderr_narrow (  
    int loglevel,  
    const wchar_t * message,  
    void * user_data)
```

Simple callback for logging to stderr in narrow (single byte) chars

Parameters

<i>loglevel</i>	a loglevel
<i>message</i>	a message

6.1.4.96 logging_callback_stderr_wide()

```
void XIMC_API logging_callback_stderr_wide (  
    int loglevel,  
    const wchar_t * message,  
    void * user_data)
```

Simple callback for logging to stderr in wide chars

Parameters

<i>loglevel</i>	a loglevel
<i>message</i>	a message

6.1.4.97 msec_sleep()

```
void XIMC_API msec_sleep (  
    unsigned int msec)
```

Sleeps for a specified amount of time

Parameters

<i>msec</i>	time in milliseconds
-------------	----------------------

6.1.4.98 open_device()

```
device_t XIMC_API open_device (  
    const char * uri)
```

Open a device with OS *uri* and return identifier of the device which can be used in calls.

Parameters

in	uri	- a device URL. Device URL has a form of "xi-com:port" or "xi-net://host/serial" or "xi-emu:///abs_path_to_file". For POSIX systems one can omit root-slash in abs_path_to_file; for example, "xi-emu:///home/user/virt_controller.bin". In case of USB-COM port, the "port" is the OS device URL. For example, "xi-com:\\\\.\\COM3" in Windows (note that double-backslash will be transformed to single-backslash) or "xi-com:///dev/ttyACM0" in Linux/Mac. In case of network device, the "host" is an IPv4 address or fully qualified domain URL (FQDN), "serial" is the device serial number in hexadecimal system. For example, "xi-net://192.168.0.1/00001234" or "xi-net://hostname.com/89ABCDEF". In case of TCP protocol, use "xi-tcp://<ip/host>:<port>". For example, "xi-tcp://192.168.0.1:1818". In case of virtual device, the "abs_file_to_file" is the full path to the virtual device's file. If it doesn't exist, then it is created and initialized with default values. For example, "xi-emu:///C:/dir/file.bin" in Windows or "xi-emu:///home/user/file.bin" in Linux/Mac.
----	-----	--

6.1.4.99 probe_device()

```
result_t XIMC_API probe_device (
    const char * uri)
```

Check if a device with OS uri *uri* is XIMC device.

Be carefully with this call because it sends some data to the device.

Parameters

in	uri	- a device uri
----	-----	----------------

6.1.4.100 reset_locks()

```
result_t XIMC_API reset_locks ()
```

Resets the error of incorrect data transmission.

6.1.4.101 service_command_updf()

```
result_t XIMC_API service_command_updf (
    device_t id)
```

The command switches the controller to update the firmware state.

Manufacturer only. After receiving this command, the firmware board sets a flag (for loader), sends an echo reply, and restarts the controller.

6.1.4.102 set_accessories_settings()

```
result_t XIMC_API set_accessories_settings (
    device_t id,
    const accessories_settings_t * accessories_settings)
```

Deprecated.

Set additional accessories' information to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>accessories_settings</i>	structure contains information about additional accessories

6.1.4.103 set_brake_settings()

```
result_t XIMC_API set_brake_settings (  
    device_t id,  
    const brake_settings_t * brake_settings)
```

Set brake control settings.

Parameters

	<i>id</i>	An identifier of a device
in	<i>brake_settings</i>	structure contains brake control settings

6.1.4.104 set_calibration_settings()

```
result_t XIMC_API set_calibration_settings (  
    device_t id,  
    const calibration_settings_t * calibration_settings)
```

Set calibration settings.

Manufacturer only. This function sends the structure with calibration settings to the controller's memory. These settings are used to convert bare ADC values to winding currents in mA and the full current in mA. Parameters are grouped into pairs, XXX_A and XXX_B, representing linear equation coefficients. The first one is the slope, the second one is the constant term. Thus, $XXX_Current[mA] = XXX_A[mA/ADC] * XXX_ADC_CODE[ADC] + XXX_B[mA]$.

See also

[calibration_settings_t](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>calibration_settings</i>	calibration settings

6.1.4.105 set_control_settings()

```
result_t XIMC_API set_control_settings (  
    device_t id,  
    const control_settings_t * control_settings)
```

Set motor control settings.

In case of CTL_MODE=1, joystick motor control is enabled. In this mode, the joystick is maximally displaced, the engine tends to move at MaxSpeed[i]. i=0 if another value hasn't been set at the previous usage. To change the speed index "i", use the buttons.

In case of CTL_MODE=2, the motor is controlled by the left/right buttons. When you click on the button, the motor starts moving in the appropriate direction at a speed MaxSpeed[0]. After Timeout[i], motor moves at speed MaxSpeed[i+1]. At the transition between MaxSpeed[i] and MaxSpeed[i+1] the motor just accelerates/decelerates as usual.

Parameters

	<i>id</i>	An identifier of a device
in	<i>control_settings</i>	structure contains motor control settings.

6.1.4.106 set_control_settings_calb()

```
result_t XIMC_API set_control_settings_calb (
    device_t id,
    const control_settings_calb_t * control_settings_calb,
    const calibration_t * calibration)
```

Set motor control settings with user units.

In case of CTL_MODE=1, joystick motor control is enabled. In this mode, while the joystick is maximally displaced, the engine tends to move at MaxSpeed[i]. i=0 if another value hasn't been set at the previous usage. To change the speed index "i", use the buttons.

In case of CTL_MODE=2, the motor is controlled by the left/right buttons. When you click on the button, the motor starts moving in the appropriate direction at a speed MaxSpeed[0]. After Timeout[i], motor moves at speed MaxSpeed[i+1]. At the transition between MaxSpeed[i] and MaxSpeed[i+1] the motor just accelerates/decelerates as usual.

Parameters

	<i>id</i>	An identifier of a device
in	<i>control_settings_calb</i>	structure contains motor control settings.
	<i>calibration</i>	user unit settings

6.1.4.107 set_controller_name()

```
result_t XIMC_API set_controller_name (
    device_t id,
    const controller_name_t * controller_name)
```

Write user's controller name and internal settings to the FRAM.

Parameters

	<i>id</i>	An identifier of a device
in	<i>controller_name</i>	structure contains the previously set user's controller name

6.1.4.108 set_correction_table()

```
result_t XIMC_API set_correction_table (
    device_t id,
    const char * namefile)
```

Command of loading a correction table from a text file.

The correction table is used for position correction in case of mechanical inaccuracies. It works for some parameters in _calb commands.

Parameters

	<i>id</i>	an identifier the device
in	<i>namefile</i>	- the file name must be either a full path or a relative path. If the file name is set to NULL, the correction table will be cleared. File format: two tab-separated columns. Column headers are strings. Data is real, the dot is a delimiter. The first column is a coordinate. The second one is the deviation caused by a mechanical error. The maximum length of a table is 100 rows. Coordinate column must be sorted in ascending order.

See also

[command_move](#)
[get_position_calb](#)
[get_position_calb_t](#)
[get_status_calb](#)
[status_calb_t](#)
[get_edges_settings_calb](#)
[set_edges_settings_calb](#)
[edges_settings_calb_t](#)

6.1.4.109 set_ctp_settings()

```
result_t XIMC_API set_ctp_settings (
    device_t id,
    const ctp_settings_t * ctp_settings)
```

Set control position settings (used with stepper motor only).

When controlling the step motor with the encoder (CTP_BASE=0), it is possible to detect the loss of steps. The controller knows the number of steps per revolution (GENG::StepsPerRev) and the encoder resolution (GFBS::IPT). When the control is enabled (CTP_ENABLED is set), the controller stores the current position in the steps of SM and the current position of the encoder. Next, the encoder position is converted into steps at each step, and if the difference between the current position in steps and the encoder position is greater than CTPMinError, the flag STATE_CTP_ERROR is set.

Alternatively, the stepper motor may be controlled with the speed sensor (CTP_BASE 1). In this mode, at the active edges of the input clock, the controller stores the current value of steps. Then, at each revolution, the controller checks how many steps have been passed. When the difference is over the CTPMinError, the STATE_CTP_ERROR flag is set.

Parameters

	<i>id</i>	An identifier of a device
in	<i>ctp_settings</i>	structure contains position control settings.

6.1.4.110 set_debug_write()

```
result_t XIMC_API set_debug_write (
    device_t id,
    const debug_write_t * debug_write)
```

Write data to firmware for debug purpose.

Manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>debug_write</i>	Debug data.

6.1.4.111 `set_edges_settings()`

```
result_t XIMC_API set_edges_settings (
    device_t id,
    const edges_settings_t * edges_settings)
```

Set border and limit switches settings.

See also

[get_edges_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>edges_settings</i>	edges settings, specify types of borders, motor behavior and electrical behavior of limit switches

6.1.4.112 `set_edges_settings_calb()`

```
result_t XIMC_API set_edges_settings_calb (
    device_t id,
    const edges_settings_calb_t * edges_settings_calb,
    const calibration_t * calibration)
```

Set border and limit switches settings in user units.

See also

[get_edges_settings_calb](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>edges_settings_calb</i>	edges settings, specify types of borders, motor behavior and electrical behavior of limit switches
	<i>calibration</i>	user unit settings

Note

Attention! Some parameters of the `edges_settings_calb` structure are corrected by the coordinate correction table.

6.1.4.113 set_emf_settings()

```
result_t XIMC_API set_emf_settings (
    device_t id,
    const emf_settings_t * emf_settings)
```

Set electromechanical coefficients.

The settings are different for different stepper motors. Please set new settings when you change the motor.

See also

[get_emf_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>emf_settings</i>	EMF settings

6.1.4.114 set_encoder_information()

```
result_t XIMC_API set_encoder_information (
    device_t id,
    const encoder_information_t * encoder_information)
```

Deprecated.

Set encoder information to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>encoder_information</i>	structure contains information about encoder

6.1.4.115 set_encoder_settings()

```
result_t XIMC_API set_encoder_settings (
    device_t id,
    const encoder_settings_t * encoder_settings)
```

Deprecated.

Set encoder settings to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>encoder_settings</i>	structure contains encoder settings

6.1.4.116 set_engine_advanced_setup()

```
result_t XIMC_API set_engine_advanced_setup (  
    device_t id,  
    const engine_advanced_setup_t * engine_advanced_setup)
```

Set engine advanced settings.

Manufacturer only.

See also

[get_engine_advanced_setup](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>engine_advanced_setup</i>	EAS settings

6.1.4.117 set_engine_settings()

```
result_t XIMC_API set_engine_settings (  
    device_t id,  
    const engine_settings_t * engine_settings)
```

Set engine settings.

This function sends a structure with a set of engine settings to the controller's memory. These settings specify the motor shaft movement algorithm, list of limitations and rated characteristics. Use it when you change the motor, encoder, positioner, etc. Please note that wrong engine settings may lead to device malfunction, which can cause irreversible damage to the board.

See also

[get_engine_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>engine_settings</i>	engine settings

6.1.4.118 set_engine_settings_calb()

```
result_t XIMC_API set_engine_settings_calb (  
    device_t id,  
    const engine_settings_calb_t * engine_settings_calb,  
    const calibration_t * calibration)
```

Set user unit engine settings.

This function sends a structure with a set of engine settings to the controller's memory. These settings specify the motor shaft movement algorithm, list of limitations and rated characteristics. Use it when you change the motor, encoder, positioner etc. Please note that wrong engine settings may lead to device malfunction, which can cause irreversible damage to the board.

See also

[get_engine_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>engine_settings_calb</i>	engine settings
	<i>calibration</i>	user unit settings

6.1.4.119 set_entype_settings()

```
result_t XIMC_API set_entype_settings (
    device_t id,
    const entype_settings_t * entype_settings)
```

Set engine type and driver type.

Parameters

	<i>id</i>	An identifier of a device
in	<i>entype_settings</i>	structure contains motor type and power driver type settings

6.1.4.120 set_extended_settings()

```
result_t XIMC_API set_extended_settings (
    device_t id,
    const extended_settings_t * extended_settings)
```

Set extended settings.

Currently, it is not in use.

See also

[get_extended_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>extended_settings</i>	EST settings

6.1.4.121 set_extio_settings()

```
result_t XIMC_API set_extio_settings (
    device_t id,
    const extio_settings_t * extio_settings)
```

Set EXTIO settings.

This function sends the structure with a set of EXTIO settings to the controller's memory. By default, input events are signaled through a rising front, and output states are signaled by a high logic state.

See also

[get_extio_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>extio_settings</i>	EXTIO settings

6.1.4.122 set_feedback_settings()

```
result_t XIMC_API set_feedback_settings (
    device_t id,
    const feedback_settings_t * feedback_settings)
```

Feedback settings.

Parameters

	<i>id</i>	An identifier of a device
in	<i>IPS</i>	number of encoder counts per shaft revolution. Range: 1..65535. The field is obsolete, it is recommended to write 0 to IPS and use the extended CountsPerTurn field. You may need to update the controller firmware to the latest version.
in	<i>FeedbackType</i>	type of feedback
in	<i>FeedbackFlags</i>	flags of feedback
in	<i>CountsPerTurn</i>	number of encoder counts per shaft revolution. Range: 1..4294967295. To use the CountsPerTurn field, write 0 in the IPS field, otherwise the value from the IPS field will be used.

6.1.4.123 set_gear_information()

```
result_t XIMC_API set_gear_information (
    device_t id,
    const gear_information_t * gear_information)
```

Deprecated.

Set gear information to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>gear_information</i>	structure contains information about step gearhead

6.1.4.124 set_gear_settings()

```
result_t XIMC_API set_gear_settings (
    device_t id,
    const gear_settings_t * gear_settings)
```

Deprecated.

Set gear settings to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>gear_settings</i>	structure contains step gearhead settings

6.1.4.125 set_hallsensor_information()

```
result_t XIMC_API set_hallsensor_information (
    device_t id,
    const hallsensor_information_t * hallsensor_information)
```

Deprecated.

Set hall sensor information to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>hallsensor_information</i>	structure contains information about hall sensor

6.1.4.126 set_hallsensor_settings()

```
result_t XIMC_API set_hallsensor_settings (
    device_t id,
    const hallsensor_settings_t * hallsensor_settings)
```

Deprecated.

Set hall sensor settings to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>hallsensor_settings</i>	structure contains hall sensor settings

6.1.4.127 set_home_settings()

```
result_t XIMC_API set_home_settings (
    device_t id,
    const home_settings_t * home_settings)
```

Set home settings.

This function sends home position structure to the controller's memory.

See also

[home_settings_t](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>home_settings</i>	calibrating position settings

6.1.4.128 set_home_settings_calb()

```
result_t XIMC_API set_home_settings_calb (
    device_t id,
    const home_settings_calb_t * home_settings_calb,
    const calibration_t * calibration)
```

Set user unit home settings.

This function sends home position structure to the controller's memory.

See also

[home_settings_calb_t](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>home_settings_calb</i>	calibrating position settings
	<i>calibration</i>	user unit settings

6.1.4.129 set_joystick_settings()

```
result_t XIMC_API set_joystick_settings (
    device_t id,
    const joystick_settings_t * joystick_settings)
```

Set joystick position.

If joystick position falls outside DeadZone limits, a movement begins. The speed is defined by the joystick's position in the range from the DeadZone limit to the maximum deviation. Joystick positions inside DeadZone limits correspond to zero speed (a "soft stop" command is issued continuously), and positions beyond Low and High limits correspond to MaxSpeed[i] or -MaxSpeed[i] (see command SCTL), where $i = 0$ by default and can be changed with the left/right buttons (see command SCTL). If the next speed in the list is zero (both integer and microstep parts), the button press is ignored. The first speed in the list shouldn't be zero. DeadZone is defined in 0.1% units. See the Joystick control section on https://doc.xisupport.com/en/8smc5-usb/8SMCn-USB/Technical_specification/Additional_features/Joystick_control.html for more information.

Parameters

	<i>id</i>	An identifier of a device
in	<i>joystick_settings</i>	structure contains joystick settings

6.1.4.130 set_logging_callback()

```
void XIMC_API set_logging_callback (
    logging_callback_t logging_callback,
    void * user_data)
```

Sets a logging callback.

Call resets a callback to default (stderr, syslog) if NULL passed.

Parameters

<i>logging_callback</i>	a callback for log messages
-------------------------	-----------------------------

6.1.4.131 set_motor_information()

```
result_t XIMC_API set_motor_information (
    device_t id,
    const motor_information_t * motor_information)
```

Deprecated.

Set motor information to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>motor_information</i>	structure contains motor information

6.1.4.132 set_motor_settings()

```
result_t XIMC_API set_motor_settings (
    device_t id,
    const motor_settings_t * motor_settings)
```

Deprecated.

Set motor settings to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>motor_settings</i>	structure contains motor information

6.1.4.133 set_move_settings()

```
result_t XIMC_API set_move_settings (
    device_t id,
    const move_settings_t * move_settings)
```

Movement settings set command (speed, acceleration, threshold, etc.).

Parameters

	<i>id</i>	An identifier of a device
in	<i>move_settings</i>	structure contains move settings: speed, acceleration, deceleration etc.

6.1.4.134 set_move_settings_calb()

```
result_t XIMC_API set_move_settings_calb (
    device_t id,
    const move_settings_calb_t * move_settings_calb,
    const calibration_t * calibration)
```

User unit movement settings set command (speed, acceleration, threshold, etc.).

Parameters

	<i>id</i>	An identifier of a device
in	<i>move_settings_calb</i>	structure contains move settings: speed, acceleration, deceleration etc.
	<i>calibration</i>	user unit settings

6.1.4.135 set_network_settings()

```
result_t XIMC_API set_network_settings (
    device_t id,
    const network_settings_t * network_settings)
```

Set network settings.

Manufacturer only. This function sets the desired network settings.

See also

net_settings_t

Parameters

<i>DHCPEnabled</i>	DHCP enabled (1) or not (0)
<i>IPv4Address[4]</i>	Array[4] with an IP address
<i>SubnetMask[4]</i>	Array[4] with a subnet mask address
<i>DefaultGateway[4]</i>	Array[4] with a default gateway address

6.1.4.136 set_nonvolatile_memory()

```
result_t XIMC_API set_nonvolatile_memory (
    device_t id,
    const nonvolatile_memory_t * nonvolatile_memory)
```

Write user data into the FRAM.

Parameters

	<i>id</i>	An identifier of a device
in	<i>nonvolatile_memory</i>	user data.

6.1.4.137 set_password_settings()

```
result_t XIMC_API set_password_settings (
    device_t id,
    const password_settings_t * password_settings)
```

Sets the password.

Manufacturer only. This function sets the user password for the device's web-page.

See also

[pwd_settings_t](#)

Parameters

<i>UserPassword[20]</i>	Password for web-page
-------------------------	-----------------------

6.1.4.138 set_pid_settings()

```
result_t XIMC_API set_pid_settings (
    device_t id,
    const pid_settings_t * pid_settings)
```

Set PID settings.

This function sends the structure with a set of PID factors to the controller's memory. These settings specify the behavior of the PID routine for the positioner. These factors are slightly different for different positioners. All boards are supplied with the standard set of PID settings in the controller's flash memory. Please use it for loading new PID settings when you change positioner. Please note that wrong PID settings lead to device malfunction.

See also

[get_pid_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>pid_settings</i>	PID settings

6.1.4.139 set_position()

```
result_t XIMC_API set_position (
    device_t id,
    const set_position_t * the_set_position)
```

Sets position in steps and microsteps for stepper motor.

Sets encoder position for all engines.

Parameters

	<i>id</i>	An identifier of a device
out	<i>the_set_position</i>	structure contains motor position.

6.1.4.140 set_position_calb()

```
result_t XIMC_API set_position_calb (
    device_t id,
    const set_position_calb_t * the_set_position_calb,
    const calibration_t * calibration)
```

Sets any position value and encoder value of all engines.

In user units.

Parameters

	<i>id</i>	An identifier of a device
out	<i>the_set_position_calb</i>	structure contains motor position.
	<i>calibration</i>	user unit settings

6.1.4.141 set_power_settings()

```
result_t XIMC_API set_power_settings (
    device_t id,
    const power_settings_t * power_settings)
```

Set settings of step motor power control.

Used with a stepper motor only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>power_settings</i>	structure contains settings of step motor power control

6.1.4.142 set_secure_settings()

```
result_t XIMC_API set_secure_settings (
    device_t id,
    const secure_settings_t * secure_settings)
```

Set protection settings.

Parameters

<i>id</i>	An identifier of a device
<i>secure_settings</i>	structure with secure data

See also

`status_t::flags`

6.1.4.143 `set_serial_number()`

```
result_t XIMC_API set_serial_number (  
    device_t id,  
    const serial_number_t * serial_number)
```

Write device serial number and hardware version to the controller's flash memory.

Along with the new serial number and hardware version, a "Key" is transmitted. The SN and hardware version are changed and saved when keys match. Can be used by the manufacturer only. Used from the loader only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>serial_number</i>	structure contains new serial number and secret key.

6.1.4.144 `set_stage_information()`

```
result_t XIMC_API set_stage_information (  
    device_t id,  
    const stage_information_t * stage_information)
```

Deprecated.

Set stage information to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>stage_information</i>	structure contains stage information

6.1.4.145 `set_stage_name()`

```
result_t XIMC_API set_stage_name (  
    device_t id,  
    const stage_name_t * stage_name)
```

Write the user's stage name to EEPROM.

Parameters

	<i>id</i>	An identifier of a device
in	<i>stage_name</i>	structure contains the previously set user's stage name.

6.1.4.146 set_stage_settings()

```
result_t XIMC_API set_stage_settings (
    device_t id,
    const stage_settings_t * stage_settings)
```

Deprecated.

Set stage settings to the EEPROM. Can be used by the manufacturer only

Parameters

	<i>id</i>	An identifier of a device
in	<i>stage_settings</i>	structure contains stage settings

6.1.4.147 set_sync_in_settings()

```
result_t XIMC_API set_sync_in_settings (
    device_t id,
    const sync_in_settings_t * sync_in_settings)
```

Set input synchronization settings.

This function sends the structure with a set of input synchronization settings that specify the behavior of input synchronization to the controller's memory. All boards are supplied with the standard set of these settings.

See also

[get_sync_in_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>sync_in_settings</i>	synchronization settings

6.1.4.148 set_sync_in_settings_calb()

```
result_t XIMC_API set_sync_in_settings_calb (
    device_t id,
    const sync_in_settings_calb_t * sync_in_settings_calb,
    const calibration_t * calibration)
```

Set input user unit synchronization settings.

This function sends the structure with a set of input synchronization settings that specify the behavior of input synchronization to the controller's memory. All boards are supplied with the standard set of these settings.

See also

[get_sync_in_settings_calb](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>sync_in_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

6.1.4.149 `set_sync_out_settings()`

```
result_t XIMC_API set_sync_out_settings (
    device_t id,
    const sync_out_settings_t * sync_out_settings)
```

Set output synchronization settings.

This function sends the structure with a set of output synchronization settings that specify the behavior of output synchronization to the controller's memory. All boards are supplied with the standard set of these settings.

See also

[get_sync_out_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>sync_out_settings</i>	synchronization settings

6.1.4.150 `set_sync_out_settings_calb()`

```
result_t XIMC_API set_sync_out_settings_calb (
    device_t id,
    const sync_out_settings_calb_t * sync_out_settings_calb,
    const calibration_t * calibration)
```

Set output user unit synchronization settings.

This function sends the structure with a set of output synchronization settings that specify the behavior of output synchronization to the controller's memory. All boards are supplied with the standard set of these settings.

See also

[get_sync_in_settings_calb](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>sync_out_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

6.1.4.151 set_uart_settings()

```
result_t XIMC_API set_uart_settings (
    device_t id,
    const uart_settings_t * uart_settings)
```

Set UART settings.

This function sends the structure with UART settings to the controller's memory.

See also

[uart_settings_t](#)

Parameters

	<i>Speed</i>	UART speed
in	<i>uart_settings</i>	UART settings

6.1.4.152 write_key()

```
result_t XIMC_API write_key (
    const char * uri,
    uint8_t * key)
```

Write controller key.

Can be used by manufacturer only

Parameters

	<i>uri</i>	a uri of device
in	<i>key</i>	protection key. Range: 0..4294967295

6.1.4.153 ximc_version()

```
void XIMC_API ximc_version (
    char * version)
```

Returns a library version

Parameters

<i>version</i>	a buffer to hold a version string, 32 bytes is enough
----------------	---

6.2 ximc.h

[Go to the documentation of this file.](#)

```

00001 #ifndef INC_XIMC_H
00002 #define INC_XIMC_H
00003
00012
00025 #if defined(_WIN32) || defined(LABVIEW64_IMPORT) || defined(LABVIEW32_IMPORT) || defined(MATLAB_IMPORT)
00026     #define XIMC_API __stdcall
00027 #else
00028     #ifdef LIBXIMC_EXPORTS
00029         #define XIMC_API __attribute__((visibility("default")))
00030     #else
00031         #define XIMC_API
00032     #endif
00033 #endif
00034
00043 #if defined(_WIN32) || defined(LABVIEW64_IMPORT) || defined(LABVIEW32_IMPORT) || defined(MATLAB_IMPORT)
00044     #define XIMC_CALLCONV __stdcall
00045 #else
00046     #define XIMC_CALLCONV
00047 #endif
00048
00057 #if defined(_WIN32) || defined(LABVIEW64_IMPORT) || defined(LABVIEW32_IMPORT) || defined(MATLAB_IMPORT)
00058 #define XIMC_RETTYPE unsigned int
00059 #else
00060 #define XIMC_RETTYPE void*
00061 #endif
00062
00063
00064 #if !defined(XIMC_NO_STDINT)
00065
00066 #if ( (defined(_MSC_VER) && (_MSC_VER < 1600)) || defined(LABVIEW64_IMPORT) || defined(LABVIEW32_IMPORT)) &&
!defined(MATLAB_IMPORT)
00067 // msvc types burden
00068 typedef __int8 int8_t;
00069 typedef __int16 int16_t;
00070 typedef __int32 int32_t;
00071 typedef __int64 int64_t;
00072 typedef unsigned __int8 uint8_t;
00073 typedef unsigned __int16 uint16_t;
00074 typedef unsigned __int32 uint32_t;
00075 typedef unsigned __int64 uint64_t;
00076 #else
00077 #include <stdint.h>
00078 #endif
00079
00080 /* labview doesn't speak C99 */
00081 #if defined(LABVIEW64_IMPORT) || defined(LABVIEW32_IMPORT)
00082 typedef unsigned __int64 ulong_t;
00083 typedef __int64 long_t;
00084 #else
00085 typedef unsigned long long ulong_t;
00086 typedef long long long_t;
00087 #endif
00088
00089 #endif
00090
00091 #include <time.h>
00092
00093 #if defined(__cplusplus)
00094 extern "C"
00095 {
00096 #endif
00097
00098
00107     typedef int device_t;
00108
00117     typedef int result_t;
00118
00127     #if defined(_WIN64) || defined(_LP64) || defined(LABVIEW64_IMPORT)
00128     typedef uint64_t device_enumeration_t;
00129     #else
00130     typedef uint32_t device_enumeration_t;
00131     #endif
00132     //typedef device_enumeration_t* pdevice_enumeration_t;
00133
00142 #define device_undefined -1
00143
00152
00161 #define result_ok 0
00162
00171 #define result_error -1
00172
00181 #define result_not_implemented -2

```

```

00182
00191 #define result_value_error -3
00192
00201 #define result_nodevice -4
00202
00204
00213
00222 #define LOGLEVEL_ERROR      0x01
00231 #define LOGLEVEL_WARNING    0x02
00240 #define LOGLEVEL_INFO      0x03
00249 #define LOGLEVEL_DEBUG      0x04
00251
00252
00296 typedef struct calibration_t
00297 {
00298     double A;
00326     unsigned int MicrostepMode;
00327 } calibration_t;
00328
00336 typedef struct device_network_information_t
00337 {
00338     uint32_t ipv4;
00339     char nodename[16];
00340
00341     uint32_t axis_state;
00342     char locker_username[16];
00343     char locker_nodename[16];
00344     time_t locked_time;
00345 } device_network_information_t;
00346
00347
00348
00350 #define LIBXIMC_VERSION 3.0.2
00352
00353
00355 #define LIBXIMC_PROTOCOL_VERSION 20.14
00357
00358
00359 /*
00360 -----
00361 BEGIN OF GENERATED struct declarations
00362 -----
00363 */
00364
00376 #define ENUMERATE_PROBE      0x01
00377 #define ENUMERATE_ALL_COM    0x02
00378 #define ENUMERATE_NETWORK    0x04
00380
00381
00398 #define MOVE_STATE_MOVING    0x01
00399 #define MOVE_STATE_TARGET_SPEED 0x02
00400 #define MOVE_STATE_ANTIPLAY    0x04
00402
00403
00419 #define EEPROM_PRECEDENCE    0x01
00421
00422
00439 #define PWR_STATE_UNKNOWN    0x00
00440 #define PWR_STATE_OFF        0x01
00441 #define PWR_STATE_NORM       0x03
00442 #define PWR_STATE_REDUCT     0x04
00443 #define PWR_STATE_MAX        0x05
00445
00446
00465 #define STATE_CONTR          0x0000003F
00466 #define STATE_ERRC          0x00000001
00467 #define STATE_ERRD          0x00000002
00468 #define STATE_ERRV          0x00000004
00469 #define STATE_EEPROM_CONNECTED 0x00000010
00470 #define STATE_IS_HOMED      0x00000020
00471 #define STATE_SECUR          0x1B3FFC0
00472 #define STATE_ALARM          0x00000040
00473 #define STATE_CTP_ERROR      0x00000080
00474 #define STATE_POWER_OVERHEAT 0x00000100
00475 #define STATE_CONTROLLER_OVERHEAT 0x00000200
00476 #define STATE_OVERLOAD_POWER_VOLTAGE 0x00000400
00477 #define STATE_OVERLOAD_POWER_CURRENT 0x00000800
00478 #define STATE_OVERLOAD_USB_VOLTAGE 0x00001000
00479 #define STATE_LOW_USB_VOLTAGE 0x00002000
00480 #define STATE_OVERLOAD_USB_CURRENT 0x00004000
00481 #define STATE_BORDERS_SWAP_MISSET 0x00008000
00482 #define STATE_LOW_POWER_VOLTAGE 0x00100000
00483 #define STATE_H_BRIDGE_FAULT 0x00200000
00484 #define STATE_WINDING_RES_MISMATCH 0x01000000
00485 #define STATE_ENCODER_FAULT 0x02000000
00486 #define STATE_ENGINE_RESPONSE_ERROR 0x08000000
00487 #define STATE_EXTIO_ALARM    0x10000000

```



```

00489
00490
00509 #define STATE_DIG_SIGNAL      0xFFFF
00510 #define STATE_RIGHT_EDGE        0x0001
00511 #define STATE_LEFT_EDGE        0x0002
00512 #define STATE_BUTTON_RIGHT     0x0004
00513 #define STATE_BUTTON_LEFT      0x0008
00514 #define STATE_GPIO_PINOUT      0x0010
00515 #define STATE_GPIO_LEVEL       0x0020
00516 #define STATE_BRAKE             0x0200
00517 #define STATE_REV_SENSOR       0x0400
00518 #define STATE_SYNC_INPUT       0x0800
00519 #define STATE_SYNC_OUTPUT      0x1000
00520 #define STATE_ENC_A            0x2000
00521 #define STATE_ENC_B            0x4000
00523
00524
00553 #define ENC_STATE_ABSENT       0x00
00554 #define ENC_STATE_UNKNOWN      0x01
00555 #define ENC_STATE_MALFUNC      0x02
00556 #define ENC_STATE_REVERS       0x03
00557 #define ENC_STATE_OK           0x04
00559
00560
00577 #define WIND_A_STATE_ABSENT    0x00
00578 #define WIND_A_STATE_UNKNOWN    0x01
00579 #define WIND_A_STATE_MALFUNC    0x02
00580 #define WIND_A_STATE_OK         0x03
00581 #define WIND_B_STATE_ABSENT     0x00
00582 #define WIND_B_STATE_UNKNOWN    0x10
00583 #define WIND_B_STATE_MALFUNC    0x20
00584 #define WIND_B_STATE_OK         0x30
00586
00587
00606 #define MVCMD_NAME_BITS        0x3F
00607 #define MVCMD_UKNWN            0x00
00608 #define MVCMD_MOVE              0x01
00609 #define MVCMD_MOVR              0x02
00610 #define MVCMD_LEFT              0x03
00611 #define MVCMD_RIGHT            0x04
00612 #define MVCMD_STOP              0x05
00613 #define MVCMD_HOME              0x06
00614 #define MVCMD_LOFT              0x07
00615 #define MVCMD_SSTP              0x08
00616 #define MVCMD_ERROR             0x40
00617 #define MVCMD_RUNNING           0x80
00619
00620
00640 #define RPM_DIV_1000           0x01
00642
00643
00663 #define ENGINE_REVERSE         0x01
00664 #define ENGINE_CURRENT_AS_RMS  0x02
00665 #define ENGINE_MAX_SPEED       0x04
00666 #define ENGINE_ANTIPLAY        0x08
00667 #define ENGINE_ACCEL_ON        0x10
00668 #define ENGINE_LIMIT_VOLT      0x20
00669 #define ENGINE_LIMIT_CURR      0x40
00670 #define ENGINE_LIMIT_RPM       0x80
00672
00673
00694 #define MICROSTEP_MODE_FULL     0x01
00695 #define MICROSTEP_MODE_FRAC_2   0x02
00696 #define MICROSTEP_MODE_FRAC_4   0x03
00697 #define MICROSTEP_MODE_FRAC_8   0x04
00698 #define MICROSTEP_MODE_FRAC_16  0x05
00699 #define MICROSTEP_MODE_FRAC_32  0x06
00700 #define MICROSTEP_MODE_FRAC_64  0x07
00701 #define MICROSTEP_MODE_FRAC_128 0x08
00702 #define MICROSTEP_MODE_FRAC_256 0x09
00704
00705
00726 #define ENGINE_TYPE_NONE        0x00
00727 #define ENGINE_TYPE_DC           0x01
00728 #define ENGINE_TYPE_2DC         0x02
00729 #define ENGINE_TYPE_STEP        0x03
00730 #define ENGINE_TYPE_TEST        0x04
00731 #define ENGINE_TYPE_BRUSHLESS   0x05
00733
00734
00755 #define DRIVER_TYPE_INTEGRATE    0x02
00756 #define DRIVER_TYPE_EXTERNAL     0x03
00758
00759
00778 #define POWER_REDUCE_ENABLED    0x01
00779 #define POWER_OFF_ENABLED       0x02
00780 #define POWER_SMOOTH_CURRENT    0x04

```

```

00782
00783
00802 #define ALARM_ON_DRIVER_OVERHEATING      0x01
00803 #define LOW_UPWR_PROTECTION                 0x02
00804 #define H_BRIDGE_ALERT                     0x04
00805 #define ALARM_ON_BORDERS_SWAP_MISSET       0x08
00806 #define ALARM_FLAGS_STICKING                0x10
00807 #define BRAKING_OVERTVOLTAGE_PROTECTION     0x20
00808 #define ALARM_WINDING_MISMATCH              0x40
00809 #define ALARM_ENGINE_RESPONSE               0x80
00811
00812
00830 #define SETPOS_IGNORE_POSITION             0x01
00831 #define SETPOS_IGNORE_ENCODER               0x02
00833
00834
00850 #define FEEDBACK_ENCODER                   0x01
00851 #define FEEDBACK_EMF                       0x04
00852 #define FEEDBACK_NONE                      0x05
00853 #define FEEDBACK_ENCODER_MEDIATED          0x06
00855
00856
00872 #define FEEDBACK_ENC_REVERSE               0x01
00873 #define FEEDBACK_ENC_ADAPTIVE_HOLDING       0x02
00874 #define FEEDBACK_ENC_FILTER_NONE           0x00
00875 #define FEEDBACK_ENC_FILTER_WEAK           0x10
00876 #define FEEDBACK_ENC_FILTER_MEDIUM         0x20
00877 #define FEEDBACK_ENC_FILTER_STRONG         0x30
00878 #define FEEDBACK_ENC_FILTER_BITS           0x30
00879 #define FEEDBACK_ENC_TYPE_AUTO              0x00
00880 #define FEEDBACK_ENC_TYPE_SINGLE_ENDED     0x40
00881 #define FEEDBACK_ENC_TYPE_DIFFERENTIAL     0x80
00882 #define FEEDBACK_ENC_TYPE_BITS             0xC0
00884
00885
00899 #define SYNCIN_ENABLED                     0x01
00900 #define SYNCIN_INVERT                      0x02
00901 #define SYNCIN_GOTO_POSITION                0x04
00903
00904
00918 #define SYNCOUT_ENABLED                    0x01
00919 #define SYNCOUT_STATE                       0x02
00920 #define SYNCOUT_INVERT                     0x04
00921 #define SYNCOUT_IN_STEPS                    0x08
00922 #define SYNCOUT_ON_START                    0x10
00923 #define SYNCOUT_ON_STOP                     0x20
00924 #define SYNCOUT_ON_PERIOD                   0x40
00926
00927
00943 #define EXTIO_SETUP_OUTPUT                 0x01
00944 #define EXTIO_SETUP_INVERT                 0x02
00946
00947
00964 #define EXTIO_SETUP_MODE_IN_BITS           0x0F
00965 #define EXTIO_SETUP_MODE_IN_NOP            0x00
00966 #define EXTIO_SETUP_MODE_IN_STOP            0x01
00967 #define EXTIO_SETUP_MODE_IN_PWOF            0x02
00968 #define EXTIO_SETUP_MODE_IN_MOVR            0x03
00969 #define EXTIO_SETUP_MODE_IN_HOME             0x04
00970 #define EXTIO_SETUP_MODE_IN_ALARM           0x05
00971 #define EXTIO_SETUP_MODE_OUT_BITS           0xF0
00972 #define EXTIO_SETUP_MODE_OUT_OFF            0x00
00973 #define EXTIO_SETUP_MODE_OUT_ON              0x10
00974 #define EXTIO_SETUP_MODE_OUT_MOVING         0x20
00975 #define EXTIO_SETUP_MODE_OUT_ALARM          0x30
00976 #define EXTIO_SETUP_MODE_OUT_MOTOR_ON       0x40
00978
00979
00999 #define BORDER_IS_ENCODER                  0x01
01000 #define BORDER_STOP_LEFT                    0x02
01001 #define BORDER_STOP_RIGHT                  0x04
01002 #define BORDERS_SWAP_MISSET_DETECTION       0x08
01004
01005
01025 #define ENDER_SWAP                         0x01
01026 #define ENDER_SW1_ACTIVE_LOW                 0x02
01027 #define ENDER_SW2_ACTIVE_LOW                 0x04
01029
01030
01050 #define BRAKE_ENABLED                       0x01
01051 #define BRAKE_ENG_PWROFF                     0x02
01053
01054
01074 #define CONTROL_MODE_BITS                  0x03
01075 #define CONTROL_MODE_OFF                     0x00
01076 #define CONTROL_MODE_JOY                     0x01
01077 #define CONTROL_MODE_LR                      0x02

```

```

01078 #define CONTROL_BTN_LEFT_PUSHED_OPEN    0x04
01079 #define CONTROL_BTN_RIGHT_PUSHED_OPEN    0x08
01081
01082
01100 #define JOY_REVERSE    0x01
01102
01103
01123 #define CTP_ENABLED    0x01
01124 #define CTP_BASE    0x02
01125 #define CTP_ALARM_ON_ERROR    0x04
01126 #define REV_SENS_INV    0x08
01127 #define CTP_ERROR_CORRECTION    0x10
01129
01130
01151 #define HOME_DIR_FIRST    0x001
01152 #define HOME_DIR_SECOND    0x002
01153 #define HOME_MV_SEC_LEN    0x004
01154 #define HOME_HALF_MV    0x008
01155 #define HOME_STOP_FIRST_BITS    0x030
01156 #define HOME_STOP_FIRST_REV    0x010
01157 #define HOME_STOP_FIRST_SYN    0x020
01158 #define HOME_STOP_FIRST_LIM    0x030
01159 #define HOME_STOP_SECOND_BITS    0x0C0
01160 #define HOME_STOP_SECOND_REV    0x040
01161 #define HOME_STOP_SECOND_SYN    0x080
01162 #define HOME_STOP_SECOND_LIM    0x0C0
01163 #define HOME_USE_FAST    0x100
01165
01166
01180 #define UART_PARITY_BITS    0x03
01181 #define UART_PARITY_BIT_EVEN    0x00
01182 #define UART_PARITY_BIT_ODD    0x01
01183 #define UART_PARITY_BIT_SPACE    0x02
01184 #define UART_PARITY_BIT_MARK    0x03
01185 #define UART_PARITY_BIT_USE    0x04
01186 #define UART_STOP_BIT    0x08
01188
01189
01203 #define MOTOR_TYPE_UNKNOWN    0x00
01204 #define MOTOR_TYPE_STEP    0x01
01205 #define MOTOR_TYPE_DC    0x02
01206 #define MOTOR_TYPE_BLD_C    0x03
01208
01209
01223 #define ENCSET_DIFFERENTIAL_OUTPUT    0x001
01224 #define ENCSET_PUSH_PULL_OUTPUT    0x004
01225 #define ENCSET_INDEX_CHANNEL_PRESENT    0x010
01226 #define ENCSET_REOLUTION_SENSOR_PRESENT    0x040
01227 #define ENCSET_REOLUTION_SENSOR_ACTIVE_HIGH    0x100
01229
01230
01244 #define MB_AVAILABLE    0x01
01245 #define MB_POWERED_HOLD    0x02
01247
01248
01262 #define TS_TYPE_BITS    0x07
01263 #define TS_TYPE_UNKNOWN    0x00
01264 #define TS_TYPE_THERMOCOUPLE    0x01
01265 #define TS_TYPE_SEMICONDUCTOR    0x02
01266 #define TS_AVAILABLE    0x08
01268
01269
01283 #define LS_ON_SW1_AVAILABLE    0x01
01284 #define LS_ON_SW2_AVAILABLE    0x02
01285 #define LS_SW1_ACTIVE_LOW    0x04
01286 #define LS_SW2_ACTIVE_LOW    0x08
01287 #define LS_SHORTED    0x10
01289
01290
01306 #define BACK_EMF_INDUCTANCE_AUTO    0x01
01307 #define BACK_EMF_RESISTANCE_AUTO    0x02
01308 #define BACK_EMF_KM_AUTO    0x04
01310
01311
01323 typedef struct
01324 {
01325     unsigned int IPS;
01326     unsigned int FeedbackType;
01327     unsigned int FeedbackFlags;
01328     unsigned int CountsPerTurn;
01329 } feedback_settings_t;
01330
01345 typedef struct
01346 {
01347     unsigned int FastHome;
01348     unsigned int uFastHome;
01349     unsigned int SlowHome;

```

```

01350         unsigned int uSlowHome;
01351         int HomeDelta;
01352         int uHomeDelta;
01353         unsigned int HomeFlags;
01354     } home_settings_t;
01355
01371     typedef struct
01372     {
01373         float FastHome;
01374         float SlowHome;
01375         float HomeDelta;
01376         unsigned int HomeFlags;
01377     } home_settings_calb_t;
01378
01390     typedef struct
01391     {
01392         unsigned int Speed;
01393         unsigned int uSpeed;
01394         unsigned int Accel;
01395         unsigned int Decel;
01396         unsigned int AntiplaySpeed;
01397         unsigned int uAntiplaySpeed;
01398         unsigned int MoveFlags;
01399     } move_settings_t;
01400
01412     typedef struct
01413     {
01414         float Speed;
01426         float Accel;
01438         float Decel;
01450         float AntiplaySpeed;
01462         unsigned int MoveFlags;
01463     } move_settings_calb_t;
01464
01481     typedef struct
01482     {
01483         unsigned int NomVoltage;
01484         unsigned int NomCurrent;
01485         unsigned int NomSpeed;
01486         unsigned int uNomSpeed;
01487         unsigned int EngineFlags;
01488         int Antiplay;
01489         unsigned int MicrostepMode;
01490         unsigned int StepsPerRev;
01491     } engine_settings_t;
01492
01510     typedef struct
01511     {
01512         unsigned int NomVoltage;
01513         unsigned int NomCurrent;
01514         float NomSpeed;
01515         unsigned int EngineFlags;
01516         float Antiplay;
01517         unsigned int MicrostepMode;
01518         unsigned int StepsPerRev;
01519     } engine_settings_calb_t;
01520
01537     typedef struct
01538     {
01539         unsigned int EngineType;
01540         unsigned int DriverType;
01541     } entype_settings_t;
01542
01554     typedef struct
01555     {
01556         unsigned int HoldCurrent;
01557         unsigned int CurrReductDelay;
01558         unsigned int PowerOffDelay;
01559         unsigned int CurrentSetTime;
01560         unsigned int PowerFlags;
01561     } power_settings_t;
01562
01574     typedef struct
01575     {
01576         unsigned int LowUpwrOff;
01577         unsigned int CriticalIpwr;
01578         unsigned int CriticalUpwr;
01579         unsigned int CriticalT;
01580         unsigned int CriticalIusb;
01581         unsigned int CriticalUusb;
01582         unsigned int MinimumUusb;
01583         unsigned int Flags;
01584     } secure_settings_t;
01585
01601     typedef struct
01602     {
01603         unsigned int BorderFlags;

```

```

01604         unsigned int EnderFlags;
01605         int LeftBorder;
01606         int uLeftBorder;
01607         int RightBorder;
01608         int uRightBorder;
01609     } edges_settings_t;
01610
01626     typedef struct
01627     {
01628         unsigned int BorderFlags;
01629         unsigned int EnderFlags;
01630         float LeftBorder;
01631         float RightBorder;
01632     } edges_settings_calb_t;
01633
01651     typedef struct
01652     {
01653         unsigned int KpU;
01654         unsigned int KiU;
01655         unsigned int KdU;
01656         float Kpf;
01657         float Kif;
01658         float Kdf;
01659     } pid_settings_t;
01660
01675     typedef struct
01676     {
01677         unsigned int SyncInFlags;
01678         unsigned int ClutterTime;
01679         int Position;
01680         int uPosition;
01681         unsigned int Speed;
01682         unsigned int uSpeed;
01683         unsigned int reserved0;
01684     } sync_in_settings_t;
01685
01699     typedef struct
01700     {
01701         unsigned int SyncInFlags;
01702         unsigned int ClutterTime;
01703         float Position;
01704         float Speed;
01705         unsigned int reserved0;
01706     } sync_in_settings_calb_t;
01707
01721     typedef struct
01722     {
01723         unsigned int SyncOutFlags;
01724         unsigned int SyncOutPulseSteps;
01725         unsigned int SyncOutPeriod;
01726         unsigned int Accuracy;
01727         unsigned int uAccuracy;
01728     } sync_out_settings_t;
01729
01744     typedef struct
01745     {
01746         unsigned int SyncOutFlags;
01747         unsigned int SyncOutPulseSteps;
01748         unsigned int SyncOutPeriod;
01749         float Accuracy;
01750     } sync_out_settings_calb_t;
01751
01768     typedef struct
01769     {
01770         unsigned int EXTIOSetupFlags;
01771         unsigned int EXTIOModeFlags;
01772     } extio_settings_t;
01773
01788     typedef struct
01789     {
01790         unsigned int t1;
01791         unsigned int t2;
01792         unsigned int t3;
01793         unsigned int t4;
01794         unsigned int BrakeFlags;
01795     } brake_settings_t;
01796
01822     typedef struct
01823     {
01824         unsigned int MaxSpeed[10];
01825         unsigned int uMaxSpeed[10];
01826         unsigned int Timeout[9];
01827         unsigned int MaxClickTime;
01828         unsigned int Flags;
01829         int DeltaPosition;
01830         int uDeltaPosition;
01831     } control_settings_t;

```

```

01832
01858     typedef struct
01859     {
01860         float MaxSpeed[10];
01861         unsigned int Timeout[9];
01862         unsigned int MaxClickTime;
01863         unsigned int Flags;
01864         float DeltaPosition;
01865     } control_settings_calb_t;
01866
01899     typedef struct
01900     {
01901         unsigned int JoyLowEnd;
01902         unsigned int JoyCenter;
01903         unsigned int JoyHighEnd;
01904         unsigned int ExpFactor;
01905         unsigned int DeadZone;
01906         unsigned int JoyFlags;
01907     } joystick_settings_t;
01908
01927     typedef struct
01928     {
01929         unsigned int CTPMinError;
01930         unsigned int CTPFlags;
01931     } ctp_settings_t;
01932
01947     typedef struct
01948     {
01949         unsigned int Speed;
01950         unsigned int UARTSetupFlags;
01951     } uart_settings_t;
01952
01966     typedef struct
01967     {
01968         unsigned int DHCPEnabled;
01969         unsigned int IPv4Address[4];
01970         unsigned int SubnetMask[4];
01971         unsigned int DefaultGateway[4];
01972     } network_settings_t;
01973
01987     typedef struct
01988     {
01989         char UserPassword[21];
01990     } password_settings_t;
01991
02006     typedef struct
02007     {
02008         float CSS1_A;
02009         float CSS1_B;
02010         float CSS2_A;
02011         float CSS2_B;
02012         float FullCurrent_A;
02013         float FullCurrent_B;
02014     } calibration_settings_t;
02015
02025     typedef struct
02026     {
02027         char ControllerName[17];
02028         unsigned int CtrlFlags;
02029     } controller_name_t;
02030
02040     typedef struct
02041     {
02042         unsigned int UserData[7];
02043     } nonvolatile_memory_t;
02044
02063     typedef struct
02064     {
02065         float L;
02066         float R;
02067         float Km;
02068         unsigned int BackEMFFlags;
02069     } emf_settings_t;
02070
02087     typedef struct
02088     {
02089         unsigned int stepcloseloop_Kv;
02090         unsigned int stepcloseloop_Kp_low;
02091         unsigned int stepcloseloop_Kp_high;
02092     } engine_advanced_setup_t;
02093
02109     typedef struct
02110     {
02111         unsigned int Param1;
02112     } extended_settings_t;
02113
02128     typedef struct

```

```

02129     {
02130         int Position;
02131         int uPosition;
02132         long_t EncPosition;
02133     } get_position_t;
02134
02148     typedef struct
02149     {
02150         float Position;
02151         long_t EncPosition;
02152     } get_position_calb_t;
02153
02166     typedef struct
02167     {
02168         int Position;
02169         int uPosition;
02170         long_t EncPosition;
02171         unsigned int PosFlags;
02172     } set_position_t;
02173
02186     typedef struct
02187     {
02188         float Position;
02189         long_t EncPosition;
02190         unsigned int PosFlags;
02191     } set_position_calb_t;
02192
02205     typedef struct
02206     {
02207         unsigned int MoveSts;
02208         unsigned int MvCmdSts;
02209         unsigned int PWRSts;
02210         unsigned int EncSts;
02211         unsigned int WindSts;
02212         int CurPosition;
02213         int uCurPosition;
02214         long_t EncPosition;
02215         int CurSpeed;
02216         int uCurSpeed;
02217         int Ipwr;
02218         int Upwr;
02219         int Iusb;
02220         int Uusb;
02221         int CurI;
02222         unsigned int Flags;
02223         unsigned int GPIOFlags;
02224         unsigned int CmdBufFreeSpace;
02225     } status_t;
02226
02239     typedef struct
02240     {
02241         unsigned int MoveSts;
02242         unsigned int MvCmdSts;
02243         unsigned int PWRSts;
02244         unsigned int EncSts;
02245         unsigned int WindSts;
02246         float CurPosition;
02247         long_t EncPosition;
02248         float CurSpeed;
02249         int Ipwr;
02250         int Upwr;
02251         int Iusb;
02252         int Uusb;
02253         int CurI;
02254         unsigned int Flags;
02255         unsigned int GPIOFlags;
02256         unsigned int CmdBufFreeSpace;
02257     } status_calb_t;
02258
02269     typedef struct
02270     {
02271         int Speed[25];
02272         int Error[25];
02273         unsigned int Length;
02274     } measurements_t;
02275
02289     typedef struct
02290     {
02291         int WindingVoltageA;
02292         int WindingVoltageB;
02293         int WindingVoltageC;
02294         int WindingCurrentA;
02295         int WindingCurrentB;
02296         int WindingCurrentC;
02297         unsigned int Pot;
02298         unsigned int Joy;
02299         int AveragedPowerRatio;

```

```

02300     } chart_data_t;
02301
02312     typedef struct
02313     {
02314         char Manufacturer[5];
02315         char ManufacturerId[3];
02316         char ProductDescription[9];
02317         unsigned int Major;
02318         unsigned int Minor;
02319         unsigned int Release;
02320     } device_information_t;
02321
02332     typedef struct
02333     {
02334         unsigned int SN;
02335         uint8_t Key[32];
02336         unsigned int Major;
02337         unsigned int Minor;
02338         unsigned int Release;
02339     } serial_number_t;
02340
02354     typedef struct
02355     {
02356         unsigned int A1Voltage_ADC;
02357         unsigned int A2Voltage_ADC;
02358         unsigned int B1Voltage_ADC;
02359         unsigned int B2Voltage_ADC;
02360         unsigned int SupVoltage_ADC;
02361         unsigned int ACurrent_ADC;
02362         unsigned int BCurrent_ADC;
02363         unsigned int FullCurrent_ADC;
02364         unsigned int Temp_ADC;
02365         unsigned int Joy_ADC;
02366         unsigned int Pot_ADC;
02367         unsigned int Enc_Check_ADC;
02368         unsigned int deprecated0;
02369         int A1Voltage;
02370         int A2Voltage;
02371         int B1Voltage;
02372         int B2Voltage;
02373         int SupVoltage;
02374         int ACurrent;
02375         int BCurrent;
02376         int FullCurrent;
02377         int Temp;
02378         int Joy;
02379         int Pot;
02380         int Enc_Check;
02381         unsigned int deprecated1[2];
02382         int R;
02383         int L;
02384     } analog_data_t;
02385
02397     typedef struct
02398     {
02399         uint8_t DebugData[128];
02400     } debug_read_t;
02401
02413     typedef struct
02414     {
02415         uint8_t DebugData[128];
02416     } debug_write_t;
02417
02427     typedef struct
02428     {
02429         char PositionerName[17];
02430     } stage_name_t;
02431
02443     typedef struct
02444     {
02445         char Manufacturer[17];
02446         char PartNumber[25];
02447     } stage_information_t;
02448
02460     typedef struct
02461     {
02462         float LeadScrewPitch;
02463         char Units[9];
02464         float MaxSpeed;
02465         float TravelRange;
02466         float SupplyVoltageMin;
02467         float SupplyVoltageMax;
02468         float MaxCurrentConsumption;
02469         float HorizontalLoadCapacity;
02470         float VerticalLoadCapacity;
02471     } stage_settings_t;
02472

```



```

02484     typedef struct
02485     {
02486         char Manufacturer[17];
02487         char PartNumber[25];
02488     } motor_information_t;
02489
02501     typedef struct
02502     {
02503         unsigned int MotorType;
02504         unsigned int ReservedField;
02505         unsigned int Poles;
02506         unsigned int Phases;
02507         float NominalVoltage;
02508         float NominalCurrent;
02509         float NominalSpeed;
02510         float NominalTorque;
02511         float NominalPower;
02512         float WindingResistance;
02513         float WindingInductance;
02514         float RotorInertia;
02515         float StallTorque;
02516         float DetentTorque;
02517         float TorqueConstant;
02518         float SpeedConstant;
02519         float SpeedTorqueGradient;
02520         float MechanicalTimeConstant;
02521         float MaxSpeed;
02522         float MaxCurrent;
02523         float MaxCurrentTime;
02524         float NoLoadCurrent;
02525         float NoLoadSpeed;
02526     } motor_settings_t;
02527
02539     typedef struct
02540     {
02541         char Manufacturer[17];
02542         char PartNumber[25];
02543     } encoder_information_t;
02544
02556     typedef struct
02557     {
02558         float MaxOperatingFrequency;
02559         float SupplyVoltageMin;
02560         float SupplyVoltageMax;
02561         float MaxCurrentConsumption;
02562         unsigned int PPR;
02563         unsigned int EncoderSettings;
02564     } encoder_settings_t;
02565
02577     typedef struct
02578     {
02579         char Manufacturer[17];
02580         char PartNumber[25];
02581     } hallsensor_information_t;
02582
02594     typedef struct
02595     {
02596         float MaxOperatingFrequency;
02597         float SupplyVoltageMin;
02598         float SupplyVoltageMax;
02599         float MaxCurrentConsumption;
02600         unsigned int PPR;
02601     } hallsensor_settings_t;
02602
02614     typedef struct
02615     {
02616         char Manufacturer[17];
02617         char PartNumber[25];
02618     } gear_information_t;
02619
02631     typedef struct
02632     {
02633         float ReductionIn;
02634         float ReductionOut;
02635         float RatedInputTorque;
02636         float RatedInputSpeed;
02637         float MaxOutputBacklash;
02638         float InputInertia;
02639         float Efficiency;
02640     } gear_settings_t;
02641
02653     typedef struct
02654     {
02655         char MagneticBrakeInfo[25];
02656         float MBRatedVoltage;
02657         float MBRatedCurrent;
02658         float MBTorque;

```

```

02659         unsigned int MBSSettings;
02660         char TemperatureSensorInfo[25];
02661         float TSMIn;
02662         float TSMMax;
02663         float TSGrad;
02664         unsigned int TSSettings;
02665         unsigned int LimitSwitchesSettings;
02666     } accessories_settings_t;
02667
02680     typedef struct
02681     {
02682         uint8_t key[16];
02683     } init_random_t;
02684
02694     typedef struct
02695     {
02696         unsigned int UniqueID0;
02697         unsigned int UniqueID1;
02698         unsigned int UniqueID2;
02699         unsigned int UniqueID3;
02700     } globally_unique_identifier_t;
02701
02702  /*
02703  -----
02704  BEGIN OF GENERATED function declarations
02705  -----
02706  */
02707
02718
02720
02739     result_t XIMC_API set_feedback_settings (device_t id, const feedback_settings_t* feedback_settings);
02740
02759     result_t XIMC_API get_feedback_settings (device_t id, feedback_settings_t* feedback_settings);
02760
02777     result_t XIMC_API set_home_settings (device_t id, const home_settings_t* home_settings);
02778
02797     result_t XIMC_API set_home_settings_calb (device_t id, const home_settings_calb_t* home_settings_calb, const
calibration_t* calibration);
02798
02815     result_t XIMC_API get_home_settings (device_t id, home_settings_t* home_settings);
02816
02835     result_t XIMC_API get_home_settings_calb (device_t id, home_settings_calb_t* home_settings_calb, const
calibration_t* calibration);
02836
02849     result_t XIMC_API set_move_settings (device_t id, const move_settings_t* move_settings);
02850
02865     result_t XIMC_API set_move_settings_calb (device_t id, const move_settings_calb_t* move_settings_calb, const
calibration_t* calibration);
02866
02879     result_t XIMC_API get_move_settings (device_t id, move_settings_t* move_settings);
02880
02895     result_t XIMC_API get_move_settings_calb (device_t id, move_settings_calb_t* move_settings_calb, const
calibration_t* calibration);
02896
02915     result_t XIMC_API set_engine_settings (device_t id, const engine_settings_t* engine_settings);
02916
02937     result_t XIMC_API set_engine_settings_calb (device_t id, const engine_settings_calb_t* engine_settings_calb,
const calibration_t* calibration);
02938
02957     result_t XIMC_API get_engine_settings (device_t id, engine_settings_t* engine_settings);
02958
02979     result_t XIMC_API get_engine_settings_calb (device_t id, engine_settings_calb_t* engine_settings_calb, const
calibration_t* calibration);
02980
02993     result_t XIMC_API set_entype_settings (device_t id, const entype_settings_t* entype_settings);
02994
03007     result_t XIMC_API get_entype_settings (device_t id, entype_settings_t* entype_settings);
03008
03022     result_t XIMC_API set_power_settings (device_t id, const power_settings_t* power_settings);
03023
03038     result_t XIMC_API get_power_settings (device_t id, power_settings_t* power_settings);
03039
03053     result_t XIMC_API set_secure_settings (device_t id, const secure_settings_t* secure_settings);
03054
03068     result_t XIMC_API get_secure_settings (device_t id, secure_settings_t* secure_settings);
03069
03084     result_t XIMC_API set_edges_settings (device_t id, const edges_settings_t* edges_settings);
03085
03110     result_t XIMC_API set_edges_settings_calb (device_t id, const edges_settings_calb_t* edges_settings_calb, const
calibration_t* calibration);
03111
03126     result_t XIMC_API get_edges_settings (device_t id, edges_settings_t* edges_settings);
03127
03152     result_t XIMC_API get_edges_settings_calb (device_t id, edges_settings_calb_t* edges_settings_calb, const
calibration_t* calibration);
03153
03172     result_t XIMC_API set_pid_settings (device_t id, const pid_settings_t* pid_settings);

```

```
03173
03191     result_t XIMC_API get_pid_settings (device_t id, pid_settings_t* pid_settings);
03192
03209     result_t XIMC_API set_sync_in_settings (device_t id, const sync_in_settings_t* sync_in_settings);
03210
03229     result_t XIMC_API set_sync_in_settings_calb (device_t id, const sync_in_settings_calb_t* sync_in_settings_calb,
const calibration_t* calibration);
03230
03247     result_t XIMC_API get_sync_in_settings (device_t id, sync_in_settings_t* sync_in_settings);
03248
03267     result_t XIMC_API get_sync_in_settings_calb (device_t id, sync_in_settings_calb_t* sync_in_settings_calb, const
calibration_t* calibration);
03268
03285     result_t XIMC_API set_sync_out_settings (device_t id, const sync_out_settings_t* sync_out_settings);
03286
03305     result_t XIMC_API set_sync_out_settings_calb (device_t id, const sync_out_settings_calb_t*
sync_out_settings_calb, const calibration_t* calibration);
03306
03321     result_t XIMC_API get_sync_out_settings (device_t id, sync_out_settings_t* sync_out_settings);
03322
03341     result_t XIMC_API get_sync_out_settings_calb (device_t id, sync_out_settings_calb_t* sync_out_settings_calb,
const calibration_t* calibration);
03342
03360     result_t XIMC_API set_extio_settings (device_t id, const extio_settings_t* extio_settings);
03361
03377     result_t XIMC_API get_extio_settings (device_t id, extio_settings_t* extio_settings);
03378
03391     result_t XIMC_API set_brake_settings (device_t id, const brake_settings_t* brake_settings);
03392
03405     result_t XIMC_API get_brake_settings (device_t id, brake_settings_t* brake_settings);
03406
03431     result_t XIMC_API set_control_settings (device_t id, const control_settings_t* control_settings);
03432
03459     result_t XIMC_API set_control_settings_calb (device_t id, const control_settings_calb_t* control_settings_calb,
const calibration_t* calibration);
03460
03485     result_t XIMC_API get_control_settings (device_t id, control_settings_t* control_settings);
03486
03513     result_t XIMC_API get_control_settings_calb (device_t id, control_settings_calb_t* control_settings_calb, const
calibration_t* calibration);
03514
03543     result_t XIMC_API set_joystick_settings (device_t id, const joystick_settings_t* joystick_settings);
03544
03573     result_t XIMC_API get_joystick_settings (device_t id, joystick_settings_t* joystick_settings);
03574
03594     result_t XIMC_API set_ctp_settings (device_t id, const ctp_settings_t* ctp_settings);
03595
03613     result_t XIMC_API get_ctp_settings (device_t id, ctp_settings_t* ctp_settings);
03614
03631     result_t XIMC_API set_uart_settings (device_t id, const uart_settings_t* uart_settings);
03632
03649     result_t XIMC_API get_uart_settings (device_t id, uart_settings_t* uart_settings);
03650
03670     result_t XIMC_API set_network_settings (device_t id, const network_settings_t* network_settings);
03671
03691     result_t XIMC_API get_network_settings (device_t id, network_settings_t* network_settings);
03692
03706     result_t XIMC_API set_password_settings (device_t id, const password_settings_t* password_settings);
03707
03721     result_t XIMC_API get_password_settings (device_t id, password_settings_t* password_settings);
03722
03739     result_t XIMC_API set_calibration_settings (device_t id, const calibration_settings_t* calibration_settings);
03740
03757     result_t XIMC_API get_calibration_settings (device_t id, calibration_settings_t* calibration_settings);
03758
03771     result_t XIMC_API set_controller_name (device_t id, const controller_name_t* controller_name);
03772
03785     result_t XIMC_API get_controller_name (device_t id, controller_name_t* controller_name);
03786
03799     result_t XIMC_API set_nonvolatile_memory (device_t id, const nonvolatile_memory_t* nonvolatile_memory);
03800
03813     result_t XIMC_API get_nonvolatile_memory (device_t id, nonvolatile_memory_t* nonvolatile_memory);
03814
03832     result_t XIMC_API set_emf_settings (device_t id, const emf_settings_t* emf_settings);
03833
03850     result_t XIMC_API get_emf_settings (device_t id, emf_settings_t* emf_settings);
03851
03870     result_t XIMC_API set_engine_advanced_setup (device_t id, const engine_advanced_setup_t* engine_advanced_setup);
03871
03890     result_t XIMC_API get_engine_advanced_setup (device_t id, engine_advanced_setup_t* engine_advanced_setup);
03891
03906     result_t XIMC_API set_extended_settings (device_t id, const extended_settings_t* extended_settings);
03907
03922     result_t XIMC_API get_extended_settings (device_t id, extended_settings_t* extended_settings);
03923
03924
03926
```

```
03937
03939
03957     result_t XIMC_API command_stop (device_t id);
03958
03973     result_t XIMC_API command_power_off (device_t id);
03974
03993     result_t XIMC_API command_move (device_t id, int Position, int uPosition);
03994
04021     result_t XIMC_API command_move_calb (device_t id, float Position, const calibration_t* calibration);
04022
04039     result_t XIMC_API command_movr (device_t id, int DeltaPosition, int uDeltaPosition);
04040
04065     result_t XIMC_API command_movr_calb (device_t id, float DeltaPosition, const calibration_t* calibration);
04066
04102     result_t XIMC_API command_home (device_t id);
04103
04114     result_t XIMC_API command_left (device_t id);
04115
04126     result_t XIMC_API command_right (device_t id);
04127
04139     result_t XIMC_API command_loft (device_t id);
04140
04151     result_t XIMC_API command_sstp (device_t id);
04152
04166     result_t XIMC_API get_position (device_t id, get_position_t* the_get_position);
04167
04190     result_t XIMC_API get_position_calb (device_t id, get_position_calb_t* the_get_position_calb, const
calibration_t* calibration);
04191
04206     result_t XIMC_API set_position (device_t id, const set_position_t* the_set_position);
04207
04222     result_t XIMC_API set_position_calb (device_t id, const set_position_calb_t* the_set_position_calb, const
calibration_t* calibration);
04223
04234     result_t XIMC_API command_zero (device_t id);
04235
04236
04238
04249
04251
04262     result_t XIMC_API command_save_settings (device_t id);
04263
04274     result_t XIMC_API command_read_settings (device_t id);
04275
04286     result_t XIMC_API command_save_robust_settings (device_t id);
04287
04298     result_t XIMC_API command_read_robust_settings (device_t id);
04299
04312     result_t XIMC_API command_eesave_settings (device_t id);
04313
04324     result_t XIMC_API command_eeread_settings (device_t id);
04325
04336     result_t XIMC_API command_start_measurements (device_t id);
04337
04356     result_t XIMC_API get_measurements (device_t id, measurements_t* measurements);
04357
04374     result_t XIMC_API get_chart_data (device_t id, chart_data_t* chart_data);
04375
04388     result_t XIMC_API get_serial_number (device_t id, unsigned int* SerialNumber);
04389
04406     result_t XIMC_API get_firmware_version (device_t id, unsigned int* Major, unsigned int* Minor, unsigned int*
Release);
04407
04418     result_t XIMC_API service_command_updf (device_t id);
04419
04420
04422
04433
04435
04454     result_t XIMC_API set_serial_number (device_t id, const serial_number_t* serial_number);
04455
04468     result_t XIMC_API get_analog_data (device_t id, analog_data_t* analog_data);
04469
04484     result_t XIMC_API get_debug_read (device_t id, debug_read_t* debug_read);
04485
04498     result_t XIMC_API set_debug_write (device_t id, const debug_write_t* debug_write);
04499
04500
04502
04513
04515
04528     result_t XIMC_API set_stage_name (device_t id, const stage_name_t* stage_name);
04529
04542     result_t XIMC_API get_stage_name (device_t id, stage_name_t* stage_name);
04543
04558     result_t XIMC_API set_stage_information (device_t id, const stage_information_t* stage_information);
04559
```

```

04572     result_t XIMC_API get_stage_information (device_t id, stage_information_t* stage_information);
04573
04588     result_t XIMC_API set_stage_settings (device_t id, const stage_settings_t* stage_settings);
04589
04602     result_t XIMC_API get_stage_settings (device_t id, stage_settings_t* stage_settings);
04603
04618     result_t XIMC_API set_motor_information (device_t id, const motor_information_t* motor_information);
04619
04632     result_t XIMC_API get_motor_information (device_t id, motor_information_t* motor_information);
04633
04648     result_t XIMC_API set_motor_settings (device_t id, const motor_settings_t* motor_settings);
04649
04662     result_t XIMC_API get_motor_settings (device_t id, motor_settings_t* motor_settings);
04663
04678     result_t XIMC_API set_encoder_information (device_t id, const encoder_information_t* encoder_information);
04679
04692     result_t XIMC_API get_encoder_information (device_t id, encoder_information_t* encoder_information);
04693
04708     result_t XIMC_API set_encoder_settings (device_t id, const encoder_settings_t* encoder_settings);
04709
04722     result_t XIMC_API get_encoder_settings (device_t id, encoder_settings_t* encoder_settings);
04723
04738     result_t XIMC_API set_hallsensor_information (device_t id, const hallsensor_information_t*
hallsensor_information);
04739
04752     result_t XIMC_API get_hallsensor_information (device_t id, hallsensor_information_t* hallsensor_information);
04753
04768     result_t XIMC_API set_hallsensor_settings (device_t id, const hallsensor_settings_t* hallsensor_settings);
04769
04782     result_t XIMC_API get_hallsensor_settings (device_t id, hallsensor_settings_t* hallsensor_settings);
04783
04798     result_t XIMC_API set_gear_information (device_t id, const gear_information_t* gear_information);
04799
04812     result_t XIMC_API get_gear_information (device_t id, gear_information_t* gear_information);
04813
04828     result_t XIMC_API set_gear_settings (device_t id, const gear_settings_t* gear_settings);
04829
04842     result_t XIMC_API get_gear_settings (device_t id, gear_settings_t* gear_settings);
04843
04858     result_t XIMC_API set_accessories_settings (device_t id, const accessories_settings_t* accessories_settings);
04859
04872     result_t XIMC_API get_accessories_settings (device_t id, accessories_settings_t* accessories_settings);
04873
04890     result_t XIMC_API get_bootloader_version (device_t id, unsigned int* Major, unsigned int* Minor, unsigned int*
Release);
04891
04904     result_t XIMC_API get_init_random (device_t id, init_random_t* init_random);
04905
04921     result_t XIMC_API get_globally_unique_identifier (device_t id, globally_unique_identifier_t*
globally_unique_identifier);
04922
04923
04924  /*
04925  -----
04926  END OF GENERATED CODE
04927  -----
04928  */
04929
04930  /* hand-crafted functions begin */
04931
04944     result_t XIMC_API goto_firmware(device_t id, uint8_t* ret);
04945
04958     result_t XIMC_API has_firmware(const char* uri, uint8_t* ret);
04959
04975     result_t XIMC_API command_update_firmware(const char* uri, const uint8_t* data, uint32_t data_size);
04976
04991     result_t XIMC_API write_key (const char* uri, uint8_t* key);
04992
05005     result_t XIMC_API command_reset(device_t id);
05006
05019     result_t XIMC_API command_clearfram(device_t id);
05020
05022
05023  // -----
05024
05036
05064     device_t XIMC_API open_device (const char* uri);
05065
05080     result_t XIMC_API close_device (device_t* id);
05081
05126     result_t XIMC_API set_correction_table(device_t id, const char* namefile);
05127
05140     result_t XIMC_API probe_device (const char* uri);
05141
05252     device_enumeration_t XIMC_API enumerate_devices(int enumerate_flags, const char *hints);
05253
05264     result_t XIMC_API free_enumerate_devices(device_enumeration_t device_enumeration);

```

```

05265
05276     int XIMC_API get_device_count(device_enumeration_t device_enumeration);
05277
05286     typedef char* pchar;
05287
05302     pchar XIMC_API get_device_name(device_enumeration_t device_enumeration, int device_index);
05303
05304
05321     result_t XIMC_API get_enumerate_device_serial(device_enumeration_t device_enumeration, int device_index,
uint32_t* serial);
05322
05339     result_t XIMC_API get_enumerate_device_information(device_enumeration_t device_enumeration, int device_index,
device_information_t* device_information);
05340
05357     result_t XIMC_API get_enumerate_device_controller_name(device_enumeration_t device_enumeration, int
device_index, controller_name_t* controller_name);
05358
05375     result_t XIMC_API get_enumerate_device_stage_name(device_enumeration_t device_enumeration, int device_index,
stage_name_t* stage_name);
05376
05393     result_t XIMC_API get_enumerate_device_network_information(device_enumeration_t device_enumeration, int
device_index, device_network_information_t* device_network_information);
05394
05402     result_t XIMC_API reset_locks ();
05403
05413     void XIMC_API msec_sleep (unsigned int msec);
05414
05424     void XIMC_API ximc_version (char* version);
05425
05426 #if !defined(MATLAB_IMPORT) && !defined(LABVIEW64_IMPORT) && !defined(LABVIEW32_IMPORT)
05427
05439     typedef void (XIMC_CALLCONV *logging_callback_t)(int loglevel, const wchar_t* message, void* user_data);
05440
05452     void XIMC_API logging_callback_stderr_wide(int loglevel, const wchar_t* message, void* user_data);
05453
05465     void XIMC_API logging_callback_stderr_narrow(int loglevel, const wchar_t* message, void* user_data);
05466
05479     void XIMC_API set_logging_callback(logging_callback_t logging_callback, void* user_data);
05480
05481 #endif
05482
05506     result_t XIMC_API get_status (device_t id, status_t* status);
05507
05533     result_t XIMC_API get_status_calb (device_t id, status_calb_t* status, const calibration_t* calibration);
05534
05560     result_t XIMC_API get_device_information (device_t id, device_information_t* device_information);
05561
05580     result_t XIMC_API command_wait_for_stop(device_t id, uint32_t refresh_interval_ms);
05581
05594     result_t XIMC_API command_homezero(device_t id);
05596
05597 #if defined(__cplusplus)
05598 };
05599 #endif
05600
05601 #endif
05602
05603 // vim: ts=4 shiftwidth=4
05604

```

Index

- A
 - calibration_t, [22](#)
- A1Voltage
 - analog_data_t, [14](#)
- A1Voltage_ADC
 - analog_data_t, [14](#)
- A2Voltage
 - analog_data_t, [14](#)
- A2Voltage_ADC
 - analog_data_t, [14](#)
- Accel
 - move_settings_calb_t, [70](#)
 - move_settings_t, [72](#)
- accessories_settings_t, [10](#)
 - LimitSwitchesSettings, [11](#)
 - MagneticBrakeInfo, [11](#)
 - MBRatedCurrent, [11](#)
 - MBRatedVoltage, [11](#)
 - MBSettings, [11](#)
 - MBTorque, [11](#)
 - TemperatureSensorInfo, [12](#)
 - TSGrad, [12](#)
 - TSMaх, [12](#)
 - TSMin, [12](#)
 - TSSettings, [12](#)
- Accuracy
 - sync_out_settings_calb_t, [98](#)
 - sync_out_settings_t, [100](#)
- ACurrent
 - analog_data_t, [14](#)
- ACurrent_ADC
 - analog_data_t, [15](#)
- ALARM_ON_DRIVER_OVERHEATING
 - ximc.h, [127](#)
- analog_data_t, [12](#)
 - A1Voltage, [14](#)
 - A1Voltage_ADC, [14](#)
 - A2Voltage, [14](#)
 - A2Voltage_ADC, [14](#)
 - ACurrent, [14](#)
 - ACurrent_ADC, [15](#)
 - B1Voltage, [15](#)
 - B1Voltage_ADC, [15](#)
 - B2Voltage, [15](#)
 - B2Voltage_ADC, [15](#)
 - BCurrent, [15](#)
 - BCurrent_ADC, [15](#)
 - Enc_Check, [16](#)
 - Enc_Check_ADC, [16](#)
 - FullCurrent, [16](#)
 - FullCurrent_ADC, [16](#)
 - Joy, [16](#)
 - Joy_ADC, [16](#)
 - L, [16](#)
 - Pot, [17](#)
 - R, [17](#)
 - SupVoltage, [17](#)
 - SupVoltage_ADC, [17](#)
 - Temp, [17](#)
 - Temp_ADC, [17](#)
- Antiplay
 - engine_settings_calb_t, [41](#)
 - engine_settings_t, [43](#)
- AntiplaySpeed
 - move_settings_calb_t, [70](#)
 - move_settings_t, [72](#)
- AveragedPowerRatio
 - chart_data_t, [23](#)
- B1Voltage
 - analog_data_t, [15](#)
- B1Voltage_ADC
 - analog_data_t, [15](#)
- B2Voltage
 - analog_data_t, [15](#)
- B2Voltage_ADC
 - analog_data_t, [15](#)
- BACK_EMF_INDUCTANCE_AUTO
 - ximc.h, [127](#)
- BACK_EMF_KM_AUTO
 - ximc.h, [127](#)
- BACK_EMF_RESISTANCE_AUTO
 - ximc.h, [127](#)
- BackEMFFlags
 - emf_settings_t, [36](#)
- BCurrent
 - analog_data_t, [15](#)
- BCurrent_ADC
 - analog_data_t, [15](#)
- BORDER_IS_ENCODER
 - ximc.h, [127](#)
- BORDER_STOP_LEFT
 - ximc.h, [127](#)
- BORDER_STOP_RIGHT
 - ximc.h, [127](#)

- BorderFlags
 - edges_settings_calb_t, [33](#)
 - edges_settings_t, [35](#)
- BORDERS_SWAP_MISSET_DETECTION
 - ximc.h, [128](#)
- BRAKE_ENABLED
 - ximc.h, [128](#)
- BRAKE_ENG_PWROFF
 - ximc.h, [128](#)
- brake_settings_t, [18](#)
 - BrakeFlags, [18](#)
 - t1, [18](#)
 - t2, [18](#)
 - t3, [19](#)
 - t4, [19](#)
- BrakeFlags
 - brake_settings_t, [18](#)
- BRAKING_OVERVOLTAGE_PROTECTION
 - ximc.h, [128](#)
- calibration_settings_t, [19](#)
 - CSS1_A, [20](#)
 - CSS1_B, [20](#)
 - CSS2_A, [20](#)
 - CSS2_B, [20](#)
 - FullCurrent_A, [20](#)
 - FullCurrent_B, [20](#)
- calibration_t, [21](#)
 - A, [22](#)
 - ximc.h, [154](#)
- chart_data_t, [22](#)
 - AveragedPowerRatio, [23](#)
 - Joy, [23](#)
 - Pot, [23](#)
 - WindingCurrentA, [23](#)
 - WindingCurrentB, [23](#)
 - WindingCurrentC, [23](#)
 - WindingVoltageA, [24](#)
 - WindingVoltageB, [24](#)
 - WindingVoltageC, [24](#)
- close_device
 - ximc.h, [155](#)
- ClutterTime
 - sync_in_settings_calb_t, [96](#)
 - sync_in_settings_t, [97](#)
- CmdBufFreeSpace
 - status_calb_t, [89](#)
 - status_t, [92](#)
- command_clear_fram
 - ximc.h, [155](#)
- command_eeread_settings
 - ximc.h, [155](#)
- command_eesave_settings
 - ximc.h, [156](#)
- command_home
 - ximc.h, [156](#)
- command_homezero
 - ximc.h, [156](#)
- command_left
 - ximc.h, [157](#)
- command_loft
 - ximc.h, [157](#)
- command_move
 - ximc.h, [157](#)
- command_move_calb
 - ximc.h, [157](#)
- command_movr
 - ximc.h, [158](#)
- command_movr_calb
 - ximc.h, [158](#)
- command_power_off
 - ximc.h, [159](#)
- command_read_robust_settings
 - ximc.h, [159](#)
- command_read_settings
 - ximc.h, [159](#)
- command_reset
 - ximc.h, [160](#)
- command_right
 - ximc.h, [160](#)
- command_save_robust_settings
 - ximc.h, [160](#)
- command_save_settings
 - ximc.h, [160](#)
- command_sstp
 - ximc.h, [161](#)
- command_start_measurements
 - ximc.h, [161](#)
- command_stop
 - ximc.h, [161](#)
- command_update_firmware
 - ximc.h, [161](#)
- command_wait_for_stop
 - ximc.h, [162](#)
- command_zero
 - ximc.h, [162](#)
- CONTROL_BTN_LEFT_PUSHED_OPEN
 - ximc.h, [128](#)
- CONTROL_BTN_RIGHT_PUSHED_OPEN
 - ximc.h, [128](#)
- CONTROL_MODE_BITS
 - ximc.h, [128](#)
- CONTROL_MODE_JOY
 - ximc.h, [129](#)
- CONTROL_MODE_LR
 - ximc.h, [129](#)
- CONTROL_MODE_OFF
 - ximc.h, [129](#)
- control_settings_calb_t, [24](#)
 - Flags, [25](#)
 - MaxClickTime, [25](#)
 - MaxSpeed, [25](#)

- Timeout, [25](#)
- control_settings_t, [26](#)
 - Flags, [26](#)
 - MaxClickTime, [26](#)
 - MaxSpeed, [27](#)
 - Timeout, [27](#)
 - uDeltaPosition, [27](#)
 - uMaxSpeed, [27](#)
- controller_name_t, [27](#)
 - ControllerName, [28](#)
 - CtrlFlags, [28](#)
- ControllerName
 - controller_name_t, [28](#)
- CountsPerTurn
 - feedback_settings_t, [48](#)
- Criticalpwr
 - secure_settings_t, [79](#)
- Criticalusb
 - secure_settings_t, [79](#)
- CriticalT
 - secure_settings_t, [79](#)
- CriticalUpwr
 - secure_settings_t, [80](#)
- CriticalUusb
 - secure_settings_t, [80](#)
- CSS1_A
 - calibration_settings_t, [20](#)
- CSS1_B
 - calibration_settings_t, [20](#)
- CSS2_A
 - calibration_settings_t, [20](#)
- CSS2_B
 - calibration_settings_t, [20](#)
- CTP_ALARM_ON_ERROR
 - ximc.h, [129](#)
- CTP_BASE
 - ximc.h, [129](#)
- CTP_ENABLED
 - ximc.h, [129](#)
- CTP_ERROR_CORRECTION
 - ximc.h, [129](#)
- ctp_settings_t, [28](#)
 - CTPFlags, [29](#)
 - CTPMinError, [29](#)
- CTPFlags
 - ctp_settings_t, [29](#)
- CTPMinError
 - ctp_settings_t, [29](#)
- CtrlFlags
 - controller_name_t, [28](#)
- CurPosition
 - status_calb_t, [89](#)
 - status_t, [92](#)
- CurrentSetTime
 - power_settings_t, [78](#)
- CurrReductDelay
 - power_settings_t, [78](#)
- CurSpeed
 - status_calb_t, [89](#)
 - status_t, [92](#)
- CurT
 - status_calb_t, [89](#)
 - status_t, [93](#)
- DeadZone
 - joystick_settings_t, [62](#)
- debug_read_t, [29](#)
 - DebugData, [30](#)
- debug_write_t, [30](#)
 - DebugData, [31](#)
- DebugData
 - debug_read_t, [30](#)
 - debug_write_t, [31](#)
- Decel
 - move_settings_calb_t, [71](#)
 - move_settings_t, [72](#)
- DefaultGateway
 - network_settings_t, [74](#)
- DetentTorque
 - motor_settings_t, [66](#)
- device_information_t, [31](#)
 - Major, [31](#)
 - Minor, [31](#)
 - Release, [32](#)
- device_network_information_t, [32](#)
 - ximc.h, [154](#)
- DHCPEnabled
 - network_settings_t, [74](#)
- DRIVER_TYPE_EXTERNAL
 - ximc.h, [130](#)
- DRIVER_TYPE_INTEGRATE
 - ximc.h, [130](#)
- DriverType
 - entype_settings_t, [45](#)
- edges_settings_calb_t, [32](#)
 - BorderFlags, [33](#)
 - EnderFlags, [33](#)
 - LeftBorder, [33](#)
 - RightBorder, [33](#)
- edges_settings_t, [34](#)
 - BorderFlags, [35](#)
 - EnderFlags, [35](#)
 - LeftBorder, [35](#)
 - RightBorder, [35](#)
 - uLeftBorder, [35](#)
 - uRightBorder, [35](#)
- EEPROM_PRECEDENCE
 - ximc.h, [130](#)
- Efficiency
 - gear_settings_t, [50](#)
- emf_settings_t, [36](#)
 - BackEMFFlags, [36](#)

- Km, [36](#)
- L, [36](#)
- R, [37](#)
- Enc_Check
 - analog_data_t, [16](#)
- Enc_Check_ADC
 - analog_data_t, [16](#)
- ENC_STATE_ABSENT
 - ximc.h, [130](#)
- ENC_STATE_MALFUNC
 - ximc.h, [130](#)
- ENC_STATE_OK
 - ximc.h, [130](#)
- ENC_STATE_REVERS
 - ximc.h, [130](#)
- ENC_STATE_UNKNOWN
 - ximc.h, [131](#)
- encoder_information_t, [37](#)
 - Manufacturer, [37](#)
 - PartNumber, [37](#)
- encoder_settings_t, [38](#)
 - EncoderSettings, [38](#)
 - MaxCurrentConsumption, [38](#)
 - MaxOperatingFrequency, [39](#)
 - SupplyVoltageMax, [39](#)
 - SupplyVoltageMin, [39](#)
- EncoderSettings
 - encoder_settings_t, [38](#)
- EncPosition
 - get_position_calb_t, [52](#)
 - get_position_t, [53](#)
 - set_position_calb_t, [82](#)
 - set_position_t, [83](#)
 - status_calb_t, [89](#)
 - status_t, [93](#)
- EncSts
 - status_calb_t, [89](#)
 - status_t, [93](#)
- ENDER_SW1_ACTIVE_LOW
 - ximc.h, [131](#)
- ENDER_SW2_ACTIVE_LOW
 - ximc.h, [131](#)
- ENDER_SWAP
 - ximc.h, [131](#)
- EnderFlags
 - edges_settings_calb_t, [33](#)
 - edges_settings_t, [35](#)
- ENGINE_ACCEL_ON
 - ximc.h, [131](#)
- engine_advanced_setup_t, [39](#)
 - stepcloseloop_Kp_high, [40](#)
 - stepcloseloop_Kp_low, [40](#)
 - stepcloseloop_Kw, [40](#)
- ENGINE_ANTIPLAY
 - ximc.h, [131](#)
- ENGINE_CURRENT_AS_RMS
 - ximc.h, [131](#)
- ENGINE_LIMIT_CURR
 - ximc.h, [132](#)
- ENGINE_LIMIT_RPM
 - ximc.h, [132](#)
- ENGINE_LIMIT_VOLT
 - ximc.h, [132](#)
- ENGINE_MAX_SPEED
 - ximc.h, [132](#)
- ENGINE_REVERSE
 - ximc.h, [132](#)
- engine_settings_calb_t, [40](#)
 - Antiplay, [41](#)
 - EngineFlags, [41](#)
 - MicrostepMode, [41](#)
 - NomCurrent, [42](#)
 - NomSpeed, [42](#)
 - NomVoltage, [42](#)
 - StepsPerRev, [42](#)
- engine_settings_t, [42](#)
 - Antiplay, [43](#)
 - EngineFlags, [43](#)
 - MicrostepMode, [44](#)
 - NomCurrent, [44](#)
 - NomSpeed, [44](#)
 - NomVoltage, [44](#)
 - StepsPerRev, [44](#)
 - uNomSpeed, [44](#)
- ENGINE_TYPE_2DC
 - ximc.h, [132](#)
- ENGINE_TYPE_BRUSHLESS
 - ximc.h, [133](#)
- ENGINE_TYPE_DC
 - ximc.h, [133](#)
- ENGINE_TYPE_NONE
 - ximc.h, [133](#)
- ENGINE_TYPE_STEP
 - ximc.h, [133](#)
- ENGINE_TYPE_TEST
 - ximc.h, [133](#)
- EngineFlags
 - engine_settings_calb_t, [41](#)
 - engine_settings_t, [43](#)
- EngineType
 - entype_settings_t, [45](#)
- entype_settings_t, [45](#)
 - DriverType, [45](#)
 - EngineType, [45](#)
- enumerate_devices
 - ximc.h, [162](#)
- ENUMERATE_PROBE
 - ximc.h, [133](#)
- Error
 - measurements_t, [63](#)
- ExpFactor
 - joystick_settings_t, [62](#)

- extended_settings_t, [46](#)
- extio_settings_t, [46](#)
 - EXTIOModeFlags, [47](#)
 - EXTIOSetupFlags, [47](#)
- EXTIO_SETUP_INVERT
 - ximc.h, [133](#)
- EXTIO_SETUP_MODE_IN_ALARM
 - ximc.h, [134](#)
- EXTIO_SETUP_MODE_IN_BITS
 - ximc.h, [134](#)
- EXTIO_SETUP_MODE_IN_HOME
 - ximc.h, [134](#)
- EXTIO_SETUP_MODE_IN_MOVR
 - ximc.h, [134](#)
- EXTIO_SETUP_MODE_IN_NOP
 - ximc.h, [134](#)
- EXTIO_SETUP_MODE_IN_PWOF
 - ximc.h, [134](#)
- EXTIO_SETUP_MODE_IN_STOP
 - ximc.h, [134](#)
- EXTIO_SETUP_MODE_OUT_ALARM
 - ximc.h, [135](#)
- EXTIO_SETUP_MODE_OUT_BITS
 - ximc.h, [135](#)
- EXTIO_SETUP_MODE_OUT_MOTOR_ON
 - ximc.h, [135](#)
- EXTIO_SETUP_MODE_OUT_MOVING
 - ximc.h, [135](#)
- EXTIO_SETUP_MODE_OUT_OFF
 - ximc.h, [135](#)
- EXTIO_SETUP_MODE_OUT_ON
 - ximc.h, [135](#)
- EXTIO_SETUP_OUTPUT
 - ximc.h, [135](#)
- EXTIOModeFlags
 - extio_settings_t, [47](#)
- EXTIOSetupFlags
 - extio_settings_t, [47](#)
- FastHome
 - home_settings_calb_t, [58](#)
 - home_settings_t, [59](#)
- FEEDBACK_EMF
 - ximc.h, [136](#)
- FEEDBACK_ENC_ADAPTIVE_HOLDING
 - ximc.h, [136](#)
- FEEDBACK_ENC_FILTER_BITS
 - ximc.h, [136](#)
- FEEDBACK_ENC_FILTER_MEDIUM
 - ximc.h, [136](#)
- FEEDBACK_ENC_FILTER_NONE
 - ximc.h, [136](#)
- FEEDBACK_ENC_FILTER_STRONG
 - ximc.h, [136](#)
- FEEDBACK_ENC_FILTER_WEAK
 - ximc.h, [136](#)
- FEEDBACK_ENC_REVERSE
 - ximc.h, [137](#)
- FEEDBACK_ENC_TYPE_AUTO
 - ximc.h, [137](#)
- FEEDBACK_ENC_TYPE_BITS
 - ximc.h, [137](#)
- FEEDBACK_ENC_TYPE_DIFFERENTIAL
 - ximc.h, [137](#)
- FEEDBACK_ENC_TYPE_SINGLE_ENDED
 - ximc.h, [137](#)
- FEEDBACK_ENCODER
 - ximc.h, [137](#)
- FEEDBACK_ENCODER_MEDIATED
 - ximc.h, [137](#)
- FEEDBACK_NONE
 - ximc.h, [138](#)
- feedback_settings_t, [47](#)
 - CountsPerTurn, [48](#)
 - FeedbackFlags, [48](#)
 - FeedbackType, [48](#)
 - IPS, [48](#)
- FeedbackFlags
 - feedback_settings_t, [48](#)
- FeedbackType
 - feedback_settings_t, [48](#)
- Flags
 - control_settings_calb_t, [25](#)
 - control_settings_t, [26](#)
 - secure_settings_t, [80](#)
 - status_calb_t, [90](#)
 - status_t, [93](#)
- free_enumerate_devices
 - ximc.h, [164](#)
- FullCurrent
 - analog_data_t, [16](#)
- FullCurrent_A
 - calibration_settings_t, [20](#)
- FullCurrent_ADC
 - analog_data_t, [16](#)
- FullCurrent_B
 - calibration_settings_t, [20](#)
- gear_information_t, [48](#)
 - Manufacturer, [49](#)
 - PartNumber, [49](#)
- gear_settings_t, [49](#)
 - Efficiency, [50](#)
 - InputInertia, [50](#)
 - MaxOutputBacklash, [50](#)
 - RatedInputSpeed, [51](#)
 - RatedInputTorque, [51](#)
 - ReductionIn, [51](#)
 - ReductionOut, [51](#)
- get_accessories_settings
 - ximc.h, [165](#)
- get_analog_data

ximc.h, [165](#)
get_bootloader_version
 ximc.h, [165](#)
get_brake_settings
 ximc.h, [165](#)
get_calibration_settings
 ximc.h, [166](#)
get_chart_data
 ximc.h, [166](#)
get_control_settings
 ximc.h, [167](#)
get_control_settings_calb
 ximc.h, [167](#)
get_controller_name
 ximc.h, [167](#)
get_ctp_settings
 ximc.h, [168](#)
get_debug_read
 ximc.h, [168](#)
get_device_count
 ximc.h, [169](#)
get_device_information
 ximc.h, [169](#)
get_device_name
 ximc.h, [169](#)
get_edges_settings
 ximc.h, [170](#)
get_edges_settings_calb
 ximc.h, [170](#)
get_emf_settings
 ximc.h, [170](#)
get_encoder_information
 ximc.h, [171](#)
get_encoder_settings
 ximc.h, [171](#)
get_engine_advanced_setup
 ximc.h, [171](#)
get_engine_settings
 ximc.h, [172](#)
get_engine_settings_calb
 ximc.h, [172](#)
get_entype_settings
 ximc.h, [173](#)
get_enumerate_device_controller_name
 ximc.h, [173](#)
get_enumerate_device_information
 ximc.h, [173](#)
get_enumerate_device_network_information
 ximc.h, [174](#)
get_enumerate_device_serial
 ximc.h, [174](#)
get_enumerate_device_stage_name
 ximc.h, [174](#)
get_extended_settings
 ximc.h, [175](#)
get_extio_settings
 ximc.h, [175](#)
get_feedback_settings
 ximc.h, [175](#)
get_firmware_version
 ximc.h, [176](#)
get_gear_information
 ximc.h, [176](#)
get_gear_settings
 ximc.h, [176](#)
get_globally_unique_identifier
 ximc.h, [177](#)
get_hallsensor_information
 ximc.h, [177](#)
get_hallsensor_settings
 ximc.h, [177](#)
get_home_settings
 ximc.h, [178](#)
get_home_settings_calb
 ximc.h, [178](#)
get_init_random
 ximc.h, [178](#)
get_joystick_settings
 ximc.h, [179](#)
get_measurements
 ximc.h, [179](#)
get_motor_information
 ximc.h, [179](#)
get_motor_settings
 ximc.h, [180](#)
get_move_settings
 ximc.h, [180](#)
get_move_settings_calb
 ximc.h, [180](#)
get_network_settings
 ximc.h, [181](#)
get_nonvolatile_memory
 ximc.h, [181](#)
get_password_settings
 ximc.h, [181](#)
get_pid_settings
 ximc.h, [182](#)
get_position
 ximc.h, [182](#)
get_position_calb
 ximc.h, [182](#)
get_position_calb_t, [51](#)
 EncPosition, [52](#)
 Position, [52](#)
get_position_t, [52](#)
 EncPosition, [53](#)
 uPosition, [53](#)
get_power_settings
 ximc.h, [183](#)
get_secure_settings
 ximc.h, [183](#)
get_serial_number

- ximc.h, [183](#)
- get_stage_information
 - ximc.h, [184](#)
- get_stage_name
 - ximc.h, [184](#)
- get_stage_settings
 - ximc.h, [184](#)
- get_status
 - ximc.h, [184](#)
- get_status_calb
 - ximc.h, [185](#)
- get_sync_in_settings
 - ximc.h, [185](#)
- get_sync_in_settings_calb
 - ximc.h, [185](#)
- get_sync_out_settings
 - ximc.h, [186](#)
- get_sync_out_settings_calb
 - ximc.h, [186](#)
- get_uart_settings
 - ximc.h, [187](#)
- globally_unique_identifier_t, [53](#)
 - UniquelD0, [54](#)
 - UniquelD1, [54](#)
 - UniquelD2, [54](#)
 - UniquelD3, [54](#)
- goto_firmware
 - ximc.h, [187](#)
- GPIOFlags
 - status_calb_t, [90](#)
 - status_t, [93](#)
- H_BRIDGE_ALERT
 - ximc.h, [138](#)
- hallsensor_information_t, [54](#)
 - Manufacturer, [55](#)
 - PartNumber, [55](#)
- hallsensor_settings_t, [55](#)
 - MaxCurrentConsumption, [56](#)
 - MaxOperatingFrequency, [56](#)
 - SupplyVoltageMax, [56](#)
 - SupplyVoltageMin, [57](#)
- has_firmware
 - ximc.h, [187](#)
- HoldCurrent
 - power_settings_t, [78](#)
- HOME_DIR_FIRST
 - ximc.h, [138](#)
- HOME_DIR_SECOND
 - ximc.h, [138](#)
- HOME_HALF_MV
 - ximc.h, [138](#)
- HOME_MV_SEC_EN
 - ximc.h, [138](#)
- home_settings_calb_t, [57](#)
 - FastHome, [58](#)
 - HomeDelta, [58](#)
 - HomeFlags, [58](#)
 - SlowHome, [58](#)
- home_settings_t, [58](#)
 - FastHome, [59](#)
 - HomeDelta, [59](#)
 - HomeFlags, [59](#)
 - SlowHome, [59](#)
 - uFastHome, [59](#)
 - uHomeDelta, [60](#)
 - uSlowHome, [60](#)
- HOME_STOP_FIRST_BITS
 - ximc.h, [138](#)
- HOME_STOP_FIRST_LIM
 - ximc.h, [139](#)
- HOME_STOP_FIRST_REV
 - ximc.h, [139](#)
- HOME_STOP_FIRST_SYN
 - ximc.h, [139](#)
- HOME_STOP_SECOND_BITS
 - ximc.h, [139](#)
- HOME_STOP_SECOND_LIM
 - ximc.h, [139](#)
- HOME_STOP_SECOND_REV
 - ximc.h, [139](#)
- HOME_STOP_SECOND_SYN
 - ximc.h, [139](#)
- HOME_USE_FAST
 - ximc.h, [140](#)
- HomeDelta
 - home_settings_calb_t, [58](#)
 - home_settings_t, [59](#)
- HomeFlags
 - home_settings_calb_t, [58](#)
 - home_settings_t, [59](#)
- HorizontalLoadCapacity
 - stage_settings_t, [86](#)
- How to use with..., [4](#)
- init_random_t, [60](#)
 - key, [61](#)
- InputInertia
 - gear_settings_t, [50](#)
- Introduction, [3](#)
- IPS
 - feedback_settings_t, [48](#)
- IPv4Address
 - network_settings_t, [74](#)
- lpwr
 - status_calb_t, [90](#)
 - status_t, [93](#)
- lusb
 - status_calb_t, [90](#)
 - status_t, [93](#)
- Joy
 - analog_data_t, [16](#)

- chart_data_t, [23](#)
- Joy_ADC
 - analog_data_t, [16](#)
- JOY_REVERSE
 - ximc.h, [140](#)
- JoyCenter
 - joystick_settings_t, [62](#)
- JoyFlags
 - joystick_settings_t, [62](#)
- JoyHighEnd
 - joystick_settings_t, [62](#)
- JoyLowEnd
 - joystick_settings_t, [62](#)
- joystick_settings_t, [61](#)
 - DeadZone, [62](#)
 - ExpFactor, [62](#)
 - JoyCenter, [62](#)
 - JoyFlags, [62](#)
 - JoyHighEnd, [62](#)
 - JoyLowEnd, [62](#)
- Key
 - serial_number_t, [81](#)
- key
 - init_random_t, [61](#)
- Km
 - emf_settings_t, [36](#)
- L
 - analog_data_t, [16](#)
 - emf_settings_t, [36](#)
- LeadScrewPitch
 - stage_settings_t, [86](#)
- LeftBorder
 - edges_settings_calb_t, [33](#)
 - edges_settings_t, [35](#)
- Length
 - measurements_t, [63](#)
- libximc library, [1](#)
- LimitSwitchesSettings
 - accessories_settings_t, [11](#)
- logging_callback_stderr_narrow
 - ximc.h, [188](#)
- logging_callback_stderr_wide
 - ximc.h, [188](#)
- logging_callback_t
 - ximc.h, [154](#)
- LOW_UPWR_PROTECTION
 - ximc.h, [140](#)
- LowUpwrOff
 - secure_settings_t, [80](#)
- MagneticBrakeInfo
 - accessories_settings_t, [11](#)
- Major
 - device_information_t, [31](#)
 - serial_number_t, [81](#)
- Manufacturer
 - encoder_information_t, [37](#)
 - gear_information_t, [49](#)
 - hallsensor_information_t, [55](#)
 - motor_information_t, [64](#)
 - stage_information_t, [84](#)
- MaxClickTime
 - control_settings_calb_t, [25](#)
 - control_settings_t, [26](#)
- MaxCurrent
 - motor_settings_t, [66](#)
- MaxCurrentConsumption
 - encoder_settings_t, [38](#)
 - hallsensor_settings_t, [56](#)
 - stage_settings_t, [86](#)
- MaxCurrentTime
 - motor_settings_t, [66](#)
- MaxOperatingFrequency
 - encoder_settings_t, [39](#)
 - hallsensor_settings_t, [56](#)
- MaxOutputBacklash
 - gear_settings_t, [50](#)
- MaxSpeed
 - control_settings_calb_t, [25](#)
 - control_settings_t, [27](#)
 - motor_settings_t, [66](#)
 - stage_settings_t, [87](#)
- MBRatedCurrent
 - accessories_settings_t, [11](#)
- MBRatedVoltage
 - accessories_settings_t, [11](#)
- MBSSettings
 - accessories_settings_t, [11](#)
- MBTorque
 - accessories_settings_t, [11](#)
- measurements_t, [63](#)
 - Error, [63](#)
 - Length, [63](#)
- MechanicalTimeConstant
 - motor_settings_t, [67](#)
- MICROSTEP_MODE_FRAC_128
 - ximc.h, [140](#)
- MICROSTEP_MODE_FRAC_16
 - ximc.h, [140](#)
- MICROSTEP_MODE_FRAC_2
 - ximc.h, [140](#)
- MICROSTEP_MODE_FRAC_256
 - ximc.h, [140](#)
- MICROSTEP_MODE_FRAC_32
 - ximc.h, [141](#)
- MICROSTEP_MODE_FRAC_4
 - ximc.h, [141](#)
- MICROSTEP_MODE_FRAC_64
 - ximc.h, [141](#)
- MICROSTEP_MODE_FRAC_8
 - ximc.h, [141](#)

- MICROSTEP_MODE_FULL
 - ximc.h, [141](#)
- MicrostepMode
 - engine_settings_calb_t, [41](#)
 - engine_settings_t, [44](#)
- MinimumUusb
 - secure_settings_t, [80](#)
- Minor
 - device_information_t, [31](#)
 - serial_number_t, [81](#)
- motor_information_t, [64](#)
 - Manufacturer, [64](#)
 - PartNumber, [64](#)
- motor_settings_t, [65](#)
 - DetentTorque, [66](#)
 - MaxCurrent, [66](#)
 - MaxCurrentTime, [66](#)
 - MaxSpeed, [66](#)
 - MechanicalTimeConstant, [67](#)
 - MotorType, [67](#)
 - NoLoadCurrent, [67](#)
 - NoLoadSpeed, [67](#)
 - NominalCurrent, [67](#)
 - NominalPower, [67](#)
 - NominalSpeed, [68](#)
 - NominalTorque, [68](#)
 - NominalVoltage, [68](#)
 - Phases, [68](#)
 - Poles, [68](#)
 - RotorInertia, [68](#)
 - SpeedConstant, [69](#)
 - SpeedTorqueGradient, [69](#)
 - StallTorque, [69](#)
 - TorqueConstant, [69](#)
 - WindingInductance, [69](#)
 - WindingResistance, [69](#)
- MotorType
 - motor_settings_t, [67](#)
- move_settings_calb_t, [70](#)
 - Accel, [70](#)
 - AntiplaySpeed, [70](#)
 - Decel, [71](#)
 - MoveFlags, [71](#)
 - Speed, [71](#)
- move_settings_t, [71](#)
 - Accel, [72](#)
 - AntiplaySpeed, [72](#)
 - Decel, [72](#)
 - MoveFlags, [73](#)
 - Speed, [73](#)
 - uAntiplaySpeed, [73](#)
 - uSpeed, [73](#)
- MOVE_STATE_ANTIPLAY
 - ximc.h, [141](#)
- MOVE_STATE_MOVING
 - ximc.h, [141](#)
- MOVE_STATE_TARGET_SPEED
 - ximc.h, [142](#)
- MoveFlags
 - move_settings_calb_t, [71](#)
 - move_settings_t, [73](#)
- MoveSts
 - status_calb_t, [90](#)
 - status_t, [94](#)
- msec_sleep
 - ximc.h, [188](#)
- MVCMD_ERROR
 - ximc.h, [142](#)
- MVCMD_HOME
 - ximc.h, [142](#)
- MVCMD_LEFT
 - ximc.h, [142](#)
- MVCMD_LOFT
 - ximc.h, [142](#)
- MVCMD_MOVE
 - ximc.h, [142](#)
- MVCMD_MOVR
 - ximc.h, [142](#)
- MVCMD_NAME_BITS
 - ximc.h, [143](#)
- MVCMD_RIGHT
 - ximc.h, [143](#)
- MVCMD_RUNNING
 - ximc.h, [143](#)
- MVCMD_SSTP
 - ximc.h, [143](#)
- MVCMD_STOP
 - ximc.h, [143](#)
- MVCMD_UKNWN
 - ximc.h, [143](#)
- MvCmdSts
 - status_calb_t, [90](#)
 - status_t, [94](#)
- network_settings_t, [73](#)
 - DefaultGateway, [74](#)
 - DHCPEnabled, [74](#)
 - IPv4Address, [74](#)
 - SubnetMask, [74](#)
- NoLoadCurrent
 - motor_settings_t, [67](#)
- NoLoadSpeed
 - motor_settings_t, [67](#)
- NomCurrent
 - engine_settings_calb_t, [42](#)
 - engine_settings_t, [44](#)
- NominalCurrent
 - motor_settings_t, [67](#)
- NominalPower
 - motor_settings_t, [67](#)
- NominalSpeed
 - motor_settings_t, [68](#)

- NominalTorque
 - motor_settings_t, [68](#)
- NominalVoltage
 - motor_settings_t, [68](#)
- NomSpeed
 - engine_settings_calb_t, [42](#)
 - engine_settings_t, [44](#)
- NomVoltage
 - engine_settings_calb_t, [42](#)
 - engine_settings_t, [44](#)
- nonvolatile_memory_t, [75](#)
 - UserData, [75](#)
- open_device
 - ximc.h, [188](#)
- PartNumber
 - encoder_information_t, [37](#)
 - gear_information_t, [49](#)
 - hallsensor_information_t, [55](#)
 - motor_information_t, [64](#)
 - stage_information_t, [84](#)
- password_settings_t, [75](#)
 - UserPassword, [76](#)
- Phases
 - motor_settings_t, [68](#)
- pid_settings_t, [76](#)
- Poles
 - motor_settings_t, [68](#)
- PosFlags
 - set_position_calb_t, [82](#)
 - set_position_t, [83](#)
- Position
 - get_position_calb_t, [52](#)
 - set_position_calb_t, [83](#)
 - sync_in_settings_calb_t, [96](#)
- PositionerName
 - stage_name_t, [85](#)
- Pot
 - analog_data_t, [17](#)
 - chart_data_t, [23](#)
- POWER_OFF_ENABLED
 - ximc.h, [143](#)
- POWER_REDUCT_ENABLED
 - ximc.h, [144](#)
- power_settings_t, [77](#)
 - CurrentSetTime, [78](#)
 - CurrReductDelay, [78](#)
 - HoldCurrent, [78](#)
 - PowerFlags, [78](#)
 - PowerOffDelay, [78](#)
- POWER_SMOOTH_CURRENT
 - ximc.h, [144](#)
- PowerFlags
 - power_settings_t, [78](#)
- PowerOffDelay
 - power_settings_t, [78](#)
- probe_device
 - ximc.h, [189](#)
- PWR_STATE_MAX
 - ximc.h, [144](#)
- PWR_STATE_NORM
 - ximc.h, [144](#)
- PWR_STATE_OFF
 - ximc.h, [144](#)
- PWR_STATE_REDUCT
 - ximc.h, [144](#)
- PWR_STATE_UNKNOWN
 - ximc.h, [144](#)
- PWRSts
 - status_calb_t, [90](#)
 - status_t, [94](#)
- R
 - analog_data_t, [17](#)
 - emf_settings_t, [37](#)
- RatedInputSpeed
 - gear_settings_t, [51](#)
- RatedInputTorque
 - gear_settings_t, [51](#)
- ReductionIn
 - gear_settings_t, [51](#)
- ReductionOut
 - gear_settings_t, [51](#)
- Release
 - device_information_t, [32](#)
 - serial_number_t, [81](#)
- reset_locks
 - ximc.h, [189](#)
- REV_SENS_INV
 - ximc.h, [145](#)
- RightBorder
 - edges_settings_calb_t, [33](#)
 - edges_settings_t, [35](#)
- RotorInertia
 - motor_settings_t, [68](#)
- RPM_DIV_1000
 - ximc.h, [145](#)
- secure_settings_t, [78](#)
 - Criticalpwr, [79](#)
 - Criticalusb, [79](#)
 - CriticalT, [79](#)
 - CriticalUpwr, [80](#)
 - CriticalUusb, [80](#)
 - Flags, [80](#)
 - LowUpwrOff, [80](#)
 - MinimumUusb, [80](#)
- serial_number_t, [80](#)
 - Key, [81](#)
 - Major, [81](#)
 - Minor, [81](#)
 - Release, [81](#)
 - SN, [82](#)

service_command_updf
ximc.h, [189](#)

set_accessories_settings
ximc.h, [189](#)

set_brake_settings
ximc.h, [190](#)

set_calibration_settings
ximc.h, [190](#)

set_control_settings
ximc.h, [190](#)

set_control_settings_calb
ximc.h, [192](#)

set_controller_name
ximc.h, [192](#)

set_correction_table
ximc.h, [192](#)

set_ctp_settings
ximc.h, [193](#)

set_debug_write
ximc.h, [193](#)

set_edges_settings
ximc.h, [194](#)

set_edges_settings_calb
ximc.h, [194](#)

set_emf_settings
ximc.h, [194](#)

set_encoder_information
ximc.h, [195](#)

set_encoder_settings
ximc.h, [195](#)

set_engine_advanced_setup
ximc.h, [195](#)

set_engine_settings
ximc.h, [196](#)

set_engine_settings_calb
ximc.h, [196](#)

set_entype_settings
ximc.h, [197](#)

set_extended_settings
ximc.h, [197](#)

set_extio_settings
ximc.h, [197](#)

set_feedback_settings
ximc.h, [198](#)

set_gear_information
ximc.h, [198](#)

set_gear_settings
ximc.h, [198](#)

set_hallsensor_information
ximc.h, [199](#)

set_hallsensor_settings
ximc.h, [199](#)

set_home_settings
ximc.h, [199](#)

set_home_settings_calb
ximc.h, [200](#)

set_joystick_settings
ximc.h, [200](#)

set_logging_callback
ximc.h, [200](#)

set_motor_information
ximc.h, [201](#)

set_motor_settings
ximc.h, [201](#)

set_move_settings
ximc.h, [201](#)

set_move_settings_calb
ximc.h, [202](#)

set_network_settings
ximc.h, [202](#)

set_nonvolatile_memory
ximc.h, [202](#)

set_password_settings
ximc.h, [203](#)

set_pid_settings
ximc.h, [203](#)

set_position
ximc.h, [203](#)

set_position_calb
ximc.h, [204](#)

set_position_calb_t, [82](#)
EncPosition, [82](#)
PosFlags, [82](#)
Position, [83](#)

set_position_t, [83](#)
EncPosition, [83](#)
PosFlags, [83](#)
uPosition, [84](#)

set_power_settings
ximc.h, [204](#)

set_secure_settings
ximc.h, [204](#)

set_serial_number
ximc.h, [205](#)

set_stage_information
ximc.h, [205](#)

set_stage_name
ximc.h, [205](#)

set_stage_settings
ximc.h, [206](#)

set_sync_in_settings
ximc.h, [206](#)

set_sync_in_settings_calb
ximc.h, [206](#)

set_sync_out_settings
ximc.h, [207](#)

set_sync_out_settings_calb
ximc.h, [207](#)

set_uart_settings
ximc.h, [207](#)

SET_POS_IGNORE_ENCODER
ximc.h, [145](#)

- SETPOS_IGNORE_POSITION
 - ximc.h, [145](#)
- SlowHome
 - home_settings_calb_t, [58](#)
 - home_settings_t, [59](#)
- SN
 - serial_number_t, [82](#)
- Speed
 - move_settings_calb_t, [71](#)
 - move_settings_t, [73](#)
 - sync_in_settings_calb_t, [96](#)
 - sync_in_settings_t, [97](#)
- SpeedConstant
 - motor_settings_t, [69](#)
- SpeedTorqueGradient
 - motor_settings_t, [69](#)
- stage_information_t, [84](#)
 - Manufacturer, [84](#)
 - PartNumber, [84](#)
- stage_name_t, [85](#)
 - PositionerName, [85](#)
- stage_settings_t, [85](#)
 - HorizontalLoadCapacity, [86](#)
 - LeadScrewPitch, [86](#)
 - MaxCurrentConsumption, [86](#)
 - MaxSpeed, [87](#)
 - SupplyVoltageMax, [87](#)
 - SupplyVoltageMin, [87](#)
 - TravelRange, [87](#)
 - Units, [87](#)
 - VerticalLoadCapacity, [87](#)
- StallTorque
 - motor_settings_t, [69](#)
- STATE_ALARM
 - ximc.h, [145](#)
- STATE_BORDERS_SWAP_MISSET
 - ximc.h, [145](#)
- STATE_BRAKE
 - ximc.h, [145](#)
- STATE_BUTTON_LEFT
 - ximc.h, [146](#)
- STATE_BUTTON_RIGHT
 - ximc.h, [146](#)
- STATE_CONTR
 - ximc.h, [146](#)
- STATE_CONTROLLER_OVERHEAT
 - ximc.h, [146](#)
- STATE_CTP_ERROR
 - ximc.h, [146](#)
- STATE_DIG_SIGNAL
 - ximc.h, [146](#)
- STATE_EEPROM_CONNECTED
 - ximc.h, [146](#)
- STATE_ENC_A
 - ximc.h, [147](#)
- STATE_ENC_B
 - ximc.h, [147](#)
- STATE_ENGINE_RESPONSE_ERROR
 - ximc.h, [147](#)
- STATE_ERRC
 - ximc.h, [147](#)
- STATE_ERRD
 - ximc.h, [147](#)
- STATE_ERRV
 - ximc.h, [147](#)
- STATE_EXTIO_ALARM
 - ximc.h, [148](#)
- STATE_GPIO_LEVEL
 - ximc.h, [148](#)
- STATE_GPIO_PINOUT
 - ximc.h, [148](#)
- STATE_IS_HOMED
 - ximc.h, [148](#)
- STATE_LEFT_EDGE
 - ximc.h, [148](#)
- STATE_LOW_USB_VOLTAGE
 - ximc.h, [148](#)
- STATE_OVERLOAD_POWER_CURRENT
 - ximc.h, [148](#)
- STATE_OVERLOAD_POWER_VOLTAGE
 - ximc.h, [149](#)
- STATE_OVERLOAD_USB_CURRENT
 - ximc.h, [149](#)
- STATE_OVERLOAD_USB_VOLTAGE
 - ximc.h, [149](#)
- STATE_POWER_OVERHEAT
 - ximc.h, [149](#)
- STATE_REV_SENSOR
 - ximc.h, [149](#)
- STATE_RIGHT_EDGE
 - ximc.h, [149](#)
- STATE_SECUR
 - ximc.h, [150](#)
- STATE_SYNC_INPUT
 - ximc.h, [150](#)
- STATE_SYNC_OUTPUT
 - ximc.h, [150](#)
- STATE_WINDING_RES_MISMATCH
 - ximc.h, [150](#)
- status_calb_t, [88](#)
 - CmdBufFreeSpace, [89](#)
 - CurPosition, [89](#)
 - CurSpeed, [89](#)
 - CurT, [89](#)
 - EncPosition, [89](#)
 - EncSts, [89](#)
 - Flags, [90](#)
 - GPIOFlags, [90](#)
 - Ipwr, [90](#)
 - Iusb, [90](#)
 - MoveSts, [90](#)
 - MvCmdSts, [90](#)

- PWRSts, [90](#)
- Upwr, [91](#)
- Uusb, [91](#)
- WindSts, [91](#)
- status_t, [91](#)
 - CmdBufFreeSpace, [92](#)
 - CurPosition, [92](#)
 - CurSpeed, [92](#)
 - CurT, [93](#)
 - EncPosition, [93](#)
 - EncSts, [93](#)
 - Flags, [93](#)
 - GPIOFlags, [93](#)
 - lpwr, [93](#)
 - lusb, [93](#)
 - MoveSts, [94](#)
 - MvCmdSts, [94](#)
 - PWRSts, [94](#)
 - uCurPosition, [94](#)
 - uCurSpeed, [94](#)
 - Upwr, [94](#)
 - Uusb, [94](#)
 - WindSts, [95](#)
- stepcloseloop_Kp_high
 - engine_advanced_setup_t, [40](#)
- stepcloseloop_Kp_low
 - engine_advanced_setup_t, [40](#)
- stepcloseloop_Kw
 - engine_advanced_setup_t, [40](#)
- StepsPerRev
 - engine_settings_calb_t, [42](#)
 - engine_settings_t, [44](#)
- SubnetMask
 - network_settings_t, [74](#)
- SupplyVoltageMax
 - encoder_settings_t, [39](#)
 - hallsensor_settings_t, [56](#)
 - stage_settings_t, [87](#)
- SupplyVoltageMin
 - encoder_settings_t, [39](#)
 - hallsensor_settings_t, [57](#)
 - stage_settings_t, [87](#)
- SupVoltage
 - analog_data_t, [17](#)
- SupVoltage_ADC
 - analog_data_t, [17](#)
- sync_in_settings_calb_t, [95](#)
 - ClutterTime, [96](#)
 - Position, [96](#)
 - Speed, [96](#)
 - SyncInFlags, [96](#)
- sync_in_settings_t, [96](#)
 - ClutterTime, [97](#)
 - Speed, [97](#)
 - SyncInFlags, [97](#)
 - uPosition, [97](#)
 - uSpeed, [97](#)
- sync_out_settings_calb_t, [98](#)
 - Accuracy, [98](#)
 - SyncOutFlags, [98](#)
 - SyncOutPeriod, [99](#)
 - SyncOutPulseSteps, [99](#)
- sync_out_settings_t, [99](#)
 - Accuracy, [100](#)
 - SyncOutFlags, [100](#)
 - SyncOutPeriod, [100](#)
 - SyncOutPulseSteps, [100](#)
 - uAccuracy, [100](#)
- SYNCIN_ENABLED
 - ximc.h, [150](#)
- SYNCIN_GOTOPOSITION
 - ximc.h, [150](#)
- SYNCIN_INVERT
 - ximc.h, [150](#)
- SyncInFlags
 - sync_in_settings_calb_t, [96](#)
 - sync_in_settings_t, [97](#)
- SYNCOUT_ENABLED
 - ximc.h, [151](#)
- SYNCOUT_IN_STEPS
 - ximc.h, [151](#)
- SYNCOUT_INVERT
 - ximc.h, [151](#)
- SYNCOUT_ONPERIOD
 - ximc.h, [151](#)
- SYNCOUT_ONSTART
 - ximc.h, [151](#)
- SYNCOUT_ONSTOP
 - ximc.h, [151](#)
- SYNCOUT_STATE
 - ximc.h, [151](#)
- SyncOutFlags
 - sync_out_settings_calb_t, [98](#)
 - sync_out_settings_t, [100](#)
- SyncOutPeriod
 - sync_out_settings_calb_t, [99](#)
 - sync_out_settings_t, [100](#)
- SyncOutPulseSteps
 - sync_out_settings_calb_t, [99](#)
 - sync_out_settings_t, [100](#)
- t1
 - brake_settings_t, [18](#)
- t2
 - brake_settings_t, [18](#)
- t3
 - brake_settings_t, [19](#)
- t4
 - brake_settings_t, [19](#)
- Temp
 - analog_data_t, [17](#)
- Temp_ADC

- analog_data_t, [17](#)
- TemperatureSensorInfo
 - accessories_settings_t, [12](#)
- Timeout
 - control_settings_calb_t, [25](#)
 - control_settings_t, [27](#)
- TorqueConstant
 - motor_settings_t, [69](#)
- TravelRange
 - stage_settings_t, [87](#)
- TSGrad
 - accessories_settings_t, [12](#)
- TSMaX
 - accessories_settings_t, [12](#)
- TSMIn
 - accessories_settings_t, [12](#)
- TSSettings
 - accessories_settings_t, [12](#)
- uAccuracy
 - sync_out_settings_t, [100](#)
- uAntiplaySpeed
 - move_settings_t, [73](#)
- UART_PARITY_BITS
 - ximc.h, [152](#)
- uart_settings_t, [101](#)
 - UARTSetupFlags, [101](#)
- UARTSetupFlags
 - uart_settings_t, [101](#)
- uCurPosition
 - status_t, [94](#)
- uCurSpeed
 - status_t, [94](#)
- uDeltaPosition
 - control_settings_t, [27](#)
- uFastHome
 - home_settings_t, [59](#)
- uHomeDelta
 - home_settings_t, [60](#)
- uLeftBorder
 - edges_settings_t, [35](#)
- uMaxSpeed
 - control_settings_t, [27](#)
- UniqueID0
 - globally_unique_identifier_t, [54](#)
- UniqueID1
 - globally_unique_identifier_t, [54](#)
- UniqueID2
 - globally_unique_identifier_t, [54](#)
- UniqueID3
 - globally_unique_identifier_t, [54](#)
- Units
 - stage_settings_t, [87](#)
- uNomSpeed
 - engine_settings_t, [44](#)
- uPosition
 - get_position_t, [53](#)
 - set_position_t, [84](#)
 - sync_in_settings_t, [97](#)
- Upwr
 - status_calb_t, [91](#)
 - status_t, [94](#)
- uRightBorder
 - edges_settings_t, [35](#)
- UserData
 - nonvolatile_memory_t, [75](#)
- UserPassword
 - password_settings_t, [76](#)
- uSlowHome
 - home_settings_t, [60](#)
- uSpeed
 - move_settings_t, [73](#)
 - sync_in_settings_t, [97](#)
- Uusb
 - status_calb_t, [91](#)
 - status_t, [94](#)
- VerticalLoadCapacity
 - stage_settings_t, [87](#)
- WIND_A_STATE_ABSENT
 - ximc.h, [152](#)
- WIND_A_STATE_MALFUNC
 - ximc.h, [152](#)
- WIND_A_STATE_OK
 - ximc.h, [152](#)
- WIND_A_STATE_UNKNOWN
 - ximc.h, [152](#)
- WIND_B_STATE_ABSENT
 - ximc.h, [152](#)
- WIND_B_STATE_MALFUNC
 - ximc.h, [152](#)
- WIND_B_STATE_OK
 - ximc.h, [153](#)
- WIND_B_STATE_UNKNOWN
 - ximc.h, [153](#)
- WindingCurrentA
 - chart_data_t, [23](#)
- WindingCurrentB
 - chart_data_t, [23](#)
- WindingCurrentC
 - chart_data_t, [23](#)
- WindingInductance
 - motor_settings_t, [69](#)
- WindingResistance
 - motor_settings_t, [69](#)
- WindingVoltageA
 - chart_data_t, [24](#)
- WindingVoltageB
 - chart_data_t, [24](#)
- WindingVoltageC
 - chart_data_t, [24](#)
- WindSts

- status_calb_t, 91
- status_t, 95
- Working with user units, 8
- write_key
 - ximc.h, 208
- ximc.h, 102, 209
 - ALARM_ON_DRIVER_OVERHEATING, 127
 - BACK_EMF_INDUCTANCE_AUTO, 127
 - BACK_EMF_KM_AUTO, 127
 - BACK_EMF_RESISTANCE_AUTO, 127
 - BORDER_IS_ENCODER, 127
 - BORDER_STOP_LEFT, 127
 - BORDER_STOP_RIGHT, 127
 - BORDERS_SWAP_MISSET_DETECTION, 128
 - BRAKE_ENABLED, 128
 - BRAKE_ENG_PWROFF, 128
 - BRAKING_OVERVOLTAGE_PROTECTION, 128
 - calibration_t, 154
 - close_device, 155
 - command_clear_fram, 155
 - command_eeread_settings, 155
 - command_eesave_settings, 156
 - command_home, 156
 - command_homezero, 156
 - command_left, 157
 - command_loft, 157
 - command_move, 157
 - command_move_calb, 157
 - command_movr, 158
 - command_movr_calb, 158
 - command_power_off, 159
 - command_read_robust_settings, 159
 - command_read_settings, 159
 - command_reset, 160
 - command_right, 160
 - command_save_robust_settings, 160
 - command_save_settings, 160
 - command_sstp, 161
 - command_start_measurements, 161
 - command_stop, 161
 - command_update_firmware, 161
 - command_wait_for_stop, 162
 - command_zero, 162
 - CONTROL_BTN_LEFT_PUSHED_OPEN, 128
 - CONTROL_BTN_RIGHT_PUSHED_OPEN, 128
 - CONTROL_MODE_BITS, 128
 - CONTROL_MODE_JOY, 129
 - CONTROL_MODE_LR, 129
 - CONTROL_MODE_OFF, 129
 - CTP_ALARM_ON_ERROR, 129
 - CTP_BASE, 129
 - CTP_ENABLED, 129
 - CTP_ERROR_CORRECTION, 129
 - device_network_information_t, 154
 - DRIVER_TYPE_EXTERNAL, 130
 - DRIVER_TYPE_INTEGRATE, 130
 - EEPROM_PRECEDENCE, 130
 - ENC_STATE_ABSENT, 130
 - ENC_STATE_MALFUNC, 130
 - ENC_STATE_OK, 130
 - ENC_STATE_REVERS, 130
 - ENC_STATE_UNKNOWN, 131
 - ENDER_SW1_ACTIVE_LOW, 131
 - ENDER_SW2_ACTIVE_LOW, 131
 - ENDER_SWAP, 131
 - ENGINE_ACCEL_ON, 131
 - ENGINE_ANTIPLAY, 131
 - ENGINE_CURRENT_AS_RMS, 131
 - ENGINE_LIMIT_CURR, 132
 - ENGINE_LIMIT_RPM, 132
 - ENGINE_LIMIT_VOLT, 132
 - ENGINE_MAX_SPEED, 132
 - ENGINE_REVERSE, 132
 - ENGINE_TYPE_2DC, 132
 - ENGINE_TYPE_BRUSHLESS, 133
 - ENGINE_TYPE_DC, 133
 - ENGINE_TYPE_NONE, 133
 - ENGINE_TYPE_STEP, 133
 - ENGINE_TYPE_TEST, 133
 - enumerate_devices, 162
 - ENUMERATE_PROBE, 133
 - EXTIO_SETUP_INVERT, 133
 - EXTIO_SETUP_MODE_IN_ALARM, 134
 - EXTIO_SETUP_MODE_IN_BITS, 134
 - EXTIO_SETUP_MODE_IN_HOME, 134
 - EXTIO_SETUP_MODE_IN_MOVR, 134
 - EXTIO_SETUP_MODE_IN_NOP, 134
 - EXTIO_SETUP_MODE_IN_PWOF, 134
 - EXTIO_SETUP_MODE_IN_STOP, 134
 - EXTIO_SETUP_MODE_OUT_ALARM, 135
 - EXTIO_SETUP_MODE_OUT_BITS, 135
 - EXTIO_SETUP_MODE_OUT_MOTOR_ON, 135
 - EXTIO_SETUP_MODE_OUT_MOVING, 135
 - EXTIO_SETUP_MODE_OUT_OFF, 135
 - EXTIO_SETUP_MODE_OUT_ON, 135
 - EXTIO_SETUP_OUTPUT, 135
 - FEEDBACK_EMF, 136
 - FEEDBACK_ENC_ADAPTIVE_HOLDING, 136
 - FEEDBACK_ENC_FILTER_BITS, 136
 - FEEDBACK_ENC_FILTER_MEDIUM, 136
 - FEEDBACK_ENC_FILTER_NONE, 136
 - FEEDBACK_ENC_FILTER_STRONG, 136
 - FEEDBACK_ENC_FILTER_WEAK, 136
 - FEEDBACK_ENC_REVERSE, 137
 - FEEDBACK_ENC_TYPE_AUTO, 137
 - FEEDBACK_ENC_TYPE_BITS, 137

- FEEDBACK_ENC_TYPE_DIFFERENTIAL, 137
- FEEDBACK_ENC_TYPE_SINGLE_ENDED, 137
- FEEDBACK_ENCODER, 137
- FEEDBACK_ENCODER_MEDIATED, 137
- FEEDBACK_NONE, 138
- free_enumerate_devices, 164
- get_accessories_settings, 165
- get_analog_data, 165
- get_bootloader_version, 165
- get_brake_settings, 165
- get_calibration_settings, 166
- get_chart_data, 166
- get_control_settings, 167
- get_control_settings_calb, 167
- get_controller_name, 167
- get_ctp_settings, 168
- get_debug_read, 168
- get_device_count, 169
- get_device_information, 169
- get_device_name, 169
- get_edges_settings, 170
- get_edges_settings_calb, 170
- get_emf_settings, 170
- get_encoder_information, 171
- get_encoder_settings, 171
- get_engine_advanced_setup, 171
- get_engine_settings, 172
- get_engine_settings_calb, 172
- get_entype_settings, 173
- get_enumerate_device_controller_name, 173
- get_enumerate_device_information, 173
- get_enumerate_device_network_information, 174
- get_enumerate_device_serial, 174
- get_enumerate_device_stage_name, 174
- get_extended_settings, 175
- get_extio_settings, 175
- get_feedback_settings, 175
- get_firmware_version, 176
- get_gear_information, 176
- get_gear_settings, 176
- get_globally_unique_identifier, 177
- get_hallsensor_information, 177
- get_hallsensor_settings, 177
- get_home_settings, 178
- get_home_settings_calb, 178
- get_init_random, 178
- get_joystick_settings, 179
- get_measurements, 179
- get_motor_information, 179
- get_motor_settings, 180
- get_move_settings, 180
- get_move_settings_calb, 180
- get_network_settings, 181
- get_nonvolatile_memory, 181
- get_password_settings, 181
- get_pid_settings, 182
- get_position, 182
- get_position_calb, 182
- get_power_settings, 183
- get_secure_settings, 183
- get_serial_number, 183
- get_stage_information, 184
- get_stage_name, 184
- get_stage_settings, 184
- get_status, 184
- get_status_calb, 185
- get_sync_in_settings, 185
- get_sync_in_settings_calb, 185
- get_sync_out_settings, 186
- get_sync_out_settings_calb, 186
- get_uart_settings, 187
- goto_firmware, 187
- H_BRIDGE_ALERT, 138
- has_firmware, 187
- HOME_DIR_FIRST, 138
- HOME_DIR_SECOND, 138
- HOME_HALF_MV, 138
- HOME_MV_SEC_EN, 138
- HOME_STOP_FIRST_BITS, 138
- HOME_STOP_FIRST_LIM, 139
- HOME_STOP_FIRST_REV, 139
- HOME_STOP_FIRST_SYN, 139
- HOME_STOP_SECOND_BITS, 139
- HOME_STOP_SECOND_LIM, 139
- HOME_STOP_SECOND_REV, 139
- HOME_STOP_SECOND_SYN, 139
- HOME_USE_FAST, 140
- JOY_REVERSE, 140
- logging_callback_stderr_narrow, 188
- logging_callback_stderr_wide, 188
- logging_callback_t, 154
- LOW_UPWR_PROTECTION, 140
- MICROSTEP_MODE_FRAC_128, 140
- MICROSTEP_MODE_FRAC_16, 140
- MICROSTEP_MODE_FRAC_2, 140
- MICROSTEP_MODE_FRAC_256, 140
- MICROSTEP_MODE_FRAC_32, 141
- MICROSTEP_MODE_FRAC_4, 141
- MICROSTEP_MODE_FRAC_64, 141
- MICROSTEP_MODE_FRAC_8, 141
- MICROSTEP_MODE_FULL, 141
- MOVE_STATE_ANTIPLAY, 141
- MOVE_STATE_MOVING, 141
- MOVE_STATE_TARGET_SPEED, 142
- msec_sleep, 188
- MVCMD_ERROR, 142
- MVCMD_HOME, 142
- MVCMD_LEFT, 142
- MVCMD_LOFT, 142
- MVCMD_MOVE, 142

- MVCMD_MOVR, [142](#)
- MVCMD_NAME_BITS, [143](#)
- MVCMD_RIGHT, [143](#)
- MVCMD_RUNNING, [143](#)
- MVCMD_SSTP, [143](#)
- MVCMD_STOP, [143](#)
- MVCMD_UKNWN, [143](#)
- open_device, [188](#)
- POWER_OFF_ENABLED, [143](#)
- POWER_REDUCT_ENABLED, [144](#)
- POWER_SMOOTH_CURRENT, [144](#)
- probe_device, [189](#)
- PWR_STATE_MAX, [144](#)
- PWR_STATE_NORM, [144](#)
- PWR_STATE_OFF, [144](#)
- PWR_STATE_REDUCT, [144](#)
- PWR_STATE_UNKNOWN, [144](#)
- reset_locks, [189](#)
- REV_SENS_INV, [145](#)
- RPM_DIV_1000, [145](#)
- service_command_updf, [189](#)
- set_accessories_settings, [189](#)
- set_brake_settings, [190](#)
- set_calibration_settings, [190](#)
- set_control_settings, [190](#)
- set_control_settings_calb, [192](#)
- set_controller_name, [192](#)
- set_correction_table, [192](#)
- set_ctp_settings, [193](#)
- set_debug_write, [193](#)
- set_edges_settings, [194](#)
- set_edges_settings_calb, [194](#)
- set_emf_settings, [194](#)
- set_encoder_information, [195](#)
- set_encoder_settings, [195](#)
- set_engine_advanced_setup, [195](#)
- set_engine_settings, [196](#)
- set_engine_settings_calb, [196](#)
- set_entype_settings, [197](#)
- set_extended_settings, [197](#)
- set_extio_settings, [197](#)
- set_feedback_settings, [198](#)
- set_gear_information, [198](#)
- set_gear_settings, [198](#)
- set_hallsensor_information, [199](#)
- set_hallsensor_settings, [199](#)
- set_home_settings, [199](#)
- set_home_settings_calb, [200](#)
- set_joystick_settings, [200](#)
- set_logging_callback, [200](#)
- set_motor_information, [201](#)
- set_motor_settings, [201](#)
- set_move_settings, [201](#)
- set_move_settings_calb, [202](#)
- set_network_settings, [202](#)
- set_nonvolatile_memory, [202](#)
- set_password_settings, [203](#)
- set_pid_settings, [203](#)
- set_position, [203](#)
- set_position_calb, [204](#)
- set_power_settings, [204](#)
- set_secure_settings, [204](#)
- set_serial_number, [205](#)
- set_stage_information, [205](#)
- set_stage_name, [205](#)
- set_stage_settings, [206](#)
- set_sync_in_settings, [206](#)
- set_sync_in_settings_calb, [206](#)
- set_sync_out_settings, [207](#)
- set_sync_out_settings_calb, [207](#)
- set_uart_settings, [207](#)
- SETPOS_IGNORE_ENCODER, [145](#)
- SETPOS_IGNORE_POSITION, [145](#)
- STATE_ALARM, [145](#)
- STATE_BORDERS_SWAP_MISSET, [145](#)
- STATE_BRAKE, [145](#)
- STATE_BUTTON_LEFT, [146](#)
- STATE_BUTTON_RIGHT, [146](#)
- STATE_CONTR, [146](#)
- STATE_CONTROLLER_OVERHEAT, [146](#)
- STATE_CTP_ERROR, [146](#)
- STATE_DIG_SIGNAL, [146](#)
- STATE_EEPROM_CONNECTED, [146](#)
- STATE_ENC_A, [147](#)
- STATE_ENC_B, [147](#)
- STATE_ENGINE_RESPONSE_ERROR, [147](#)
- STATE_ERRC, [147](#)
- STATE_ERRD, [147](#)
- STATE_ERRV, [147](#)
- STATE_EXTIO_ALARM, [148](#)
- STATE_GPIO_LEVEL, [148](#)
- STATE_GPIO_PINOUT, [148](#)
- STATE_IS_HOMED, [148](#)
- STATE_LEFT_EDGE, [148](#)
- STATE_LOW_USB_VOLTAGE, [148](#)
- STATE_OVERLOAD_POWER_CURRENT, [148](#)
- STATE_OVERLOAD_POWER_VOLTAGE, [149](#)
- STATE_OVERLOAD_USB_CURRENT, [149](#)
- STATE_OVERLOAD_USB_VOLTAGE, [149](#)
- STATE_POWER_OVERHEAT, [149](#)
- STATE_REV_SENSOR, [149](#)
- STATE_RIGHT_EDGE, [149](#)
- STATE_SECUR, [150](#)
- STATE_SYNC_INPUT, [150](#)
- STATE_SYNC_OUTPUT, [150](#)
- STATE_WINDING_RES_MISMATCH, [150](#)
- SYNCIN_ENABLED, [150](#)
- SYNCIN_GOTOPOSITION, [150](#)
- SYNCIN_INVERT, [150](#)
- SYNCOUT_ENABLED, [151](#)
- SYNCOUT_IN_STEPS, [151](#)

- SYNCOUT_INVERT, [151](#)
- SYNCOUT_ONPERIOD, [151](#)
- SYNCOUT_ONSTART, [151](#)
- SYNCOUT_ONSTOP, [151](#)
- SYNCOUT_STATE, [151](#)
- UART_PARITY_BITS, [152](#)
- WIND_A_STATE_ABSENT, [152](#)
- WIND_A_STATE_MALFUNC, [152](#)
- WIND_A_STATE_OK, [152](#)
- WIND_A_STATE_UNKNOWN, [152](#)
- WIND_B_STATE_ABSENT, [152](#)
- WIND_B_STATE_MALFUNC, [152](#)
- WIND_B_STATE_OK, [153](#)
- WIND_B_STATE_UNKNOWN, [153](#)
- write_key, [208](#)
- XIMC_API, [153](#)
- XIMC_CALLCONV, [153](#)
- XIMC_RETTYPE, [153](#)
- ximc_version, [208](#)
- XIMC_API
 - ximc.h, [153](#)
- XIMC_CALLCONV
 - ximc.h, [153](#)
- XIMC_RETTYPE
 - ximc.h, [153](#)
- ximc_version
 - ximc.h, [208](#)