

libximc

2.13.6

Generated by Doxygen 1.8.2

Mon Jun 20 2022 21:56:37

# Contents

<b>1</b>	<b>libximc library</b>	<b>1</b>
1.1	What the controller does.	1
1.2	What can do libximc library	1
1.3	Assistance.	2
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	About library	3
2.2	System requirements	3
2.2.1	For rebuilding library	3
2.2.2	For using library	4
<b>3</b>	<b>How to rebuild library</b>	<b>5</b>
3.1	Building on generic UNIX	5
3.2	Building on debian-based linux systems	5
3.3	Building on redhat-based linux systems	5
3.4	Buliding on Mac OS X	6
3.5	Buliding on Windows	6
3.6	Source code access	6
<b>4</b>	<b>How to use with...</b>	<b>7</b>
4.1	Usage with C	7
4.1.1	Visual C++	7
4.1.2	CodeBlocks	7
4.1.3	MinGW	7
4.1.4	C++ Builder	8
4.1.5	XCode	8
4.1.6	GCC	8
4.2	.NET	9
4.3	Delphi	9
4.4	Java	9
4.5	Python	10

4.6	MATLAB	10
4.7	Generic logging facility	11
4.8	Required permissions	11
4.9	C-profiles	11
<b>5</b>	<b>Working with custom units</b>	<b>12</b>
5.1	The structure of the conversion units calibration_t	12
5.2	Alternative functions for working with custom units and data structures for them	12
5.3	Coordinate correction table for more accurate positioning	13
<b>6</b>	<b>Data Structure Documentation</b>	<b>14</b>
6.1	accessories_settings_t Struct Reference	14
6.1.1	Detailed Description	14
6.1.2	Field Documentation	15
6.1.2.1	LimitSwitchesSettings	15
6.1.2.2	MagneticBrakeInfo	15
6.1.2.3	MBRatedCurrent	15
6.1.2.4	MBRatedVoltage	15
6.1.2.5	MBSettings	15
6.1.2.6	MBTorque	15
6.1.2.7	TemperatureSensorInfo	15
6.1.2.8	TSGrad	15
6.1.2.9	TSMax	15
6.1.2.10	TSTMin	16
6.1.2.11	TSSettings	16
6.2	analog_data_t Struct Reference	16
6.2.1	Detailed Description	17
6.2.2	Field Documentation	17
6.2.2.1	A1Voltage	17
6.2.2.2	A1Voltage_ADC	17
6.2.2.3	A2Voltage	17
6.2.2.4	A2Voltage_ADC	18
6.2.2.5	ACurrent	18
6.2.2.6	ACurrent_ADC	18
6.2.2.7	B1Voltage	18
6.2.2.8	B1Voltage_ADC	18
6.2.2.9	B2Voltage	18
6.2.2.10	B2Voltage_ADC	18
6.2.2.11	BCurrent	18

6.2.2.12	BCurrent_ADC	18
6.2.2.13	FullCurrent	18
6.2.2.14	FullCurrent_ADC	18
6.2.2.15	H5	18
6.2.2.16	Joy	19
6.2.2.17	Joy_ADC	19
6.2.2.18	L	19
6.2.2.19	L5	19
6.2.2.20	L5_ADC	19
6.2.2.21	Pot	19
6.2.2.22	R	19
6.2.2.23	SupVoltage	19
6.2.2.24	SupVoltage_ADC	19
6.2.2.25	Temp	19
6.2.2.26	Temp_ADC	19
6.3	brake_settings_t Struct Reference	20
6.3.1	Detailed Description	20
6.3.2	Field Documentation	20
6.3.2.1	BrakeFlags	20
6.3.2.2	t1	20
6.3.2.3	t2	20
6.3.2.4	t3	20
6.3.2.5	t4	20
6.4	calibration_settings_t Struct Reference	21
6.4.1	Detailed Description	21
6.4.2	Field Documentation	21
6.4.2.1	CSS1_A	21
6.4.2.2	CSS1_B	21
6.4.2.3	CSS2_A	21
6.4.2.4	CSS2_B	21
6.4.2.5	FullCurrent_A	22
6.4.2.6	FullCurrent_B	22
6.5	calibration_t Struct Reference	22
6.5.1	Detailed Description	22
6.6	chart_data_t Struct Reference	22
6.6.1	Detailed Description	23
6.6.2	Field Documentation	23
6.6.2.1	DutyCycle	23
6.6.2.2	Joy	23

6.6.2.3	Pot . . . . .	23
6.6.2.4	WindingCurrentA . . . . .	23
6.6.2.5	WindingCurrentB . . . . .	23
6.6.2.6	WindingCurrentC . . . . .	23
6.6.2.7	WindingVoltageA . . . . .	24
6.6.2.8	WindingVoltageB . . . . .	24
6.6.2.9	WindingVoltageC . . . . .	24
6.7	control_settings_calb_t Struct Reference . . . . .	24
6.7.1	Detailed Description . . . . .	24
6.7.2	Field Documentation . . . . .	25
6.7.2.1	Flags . . . . .	25
6.7.2.2	MaxClickTime . . . . .	25
6.7.2.3	MaxSpeed . . . . .	25
6.7.2.4	Timeout . . . . .	25
6.8	control_settings_t Struct Reference . . . . .	25
6.8.1	Detailed Description . . . . .	25
6.8.2	Field Documentation . . . . .	26
6.8.2.1	Flags . . . . .	26
6.8.2.2	MaxClickTime . . . . .	26
6.8.2.3	MaxSpeed . . . . .	26
6.8.2.4	Timeout . . . . .	26
6.8.2.5	uDeltaPosition . . . . .	26
6.8.2.6	uMaxSpeed . . . . .	26
6.9	controller_name_t Struct Reference . . . . .	26
6.9.1	Detailed Description . . . . .	27
6.9.2	Field Documentation . . . . .	27
6.9.2.1	ControllerName . . . . .	27
6.9.2.2	CtrlFlags . . . . .	27
6.10	ctp_settings_t Struct Reference . . . . .	27
6.10.1	Detailed Description . . . . .	27
6.10.2	Field Documentation . . . . .	28
6.10.2.1	CTPFlags . . . . .	28
6.10.2.2	CTPMinError . . . . .	28
6.11	debug_read_t Struct Reference . . . . .	28
6.11.1	Detailed Description . . . . .	28
6.11.2	Field Documentation . . . . .	28
6.11.2.1	DebugData . . . . .	28
6.12	debug_write_t Struct Reference . . . . .	28
6.12.1	Detailed Description . . . . .	29

6.12.2 Field Documentation . . . . .	29
6.12.2.1 DebugData . . . . .	29
6.13 device_information_t Struct Reference . . . . .	29
6.13.1 Detailed Description . . . . .	29
6.13.2 Field Documentation . . . . .	29
6.13.2.1 Major . . . . .	29
6.13.2.2 Minor . . . . .	30
6.13.2.3 Release . . . . .	30
6.14 device_network_information_t Struct Reference . . . . .	30
6.14.1 Detailed Description . . . . .	30
6.15 edges_settings_calb_t Struct Reference . . . . .	30
6.15.1 Detailed Description . . . . .	31
6.15.2 Field Documentation . . . . .	31
6.15.2.1 BorderFlags . . . . .	31
6.15.2.2 EnderFlags . . . . .	31
6.15.2.3 LeftBorder . . . . .	31
6.15.2.4 RightBorder . . . . .	31
6.16 edges_settings_t Struct Reference . . . . .	31
6.16.1 Detailed Description . . . . .	32
6.16.2 Field Documentation . . . . .	32
6.16.2.1 BorderFlags . . . . .	32
6.16.2.2 EnderFlags . . . . .	32
6.16.2.3 LeftBorder . . . . .	32
6.16.2.4 RightBorder . . . . .	32
6.16.2.5 uLeftBorder . . . . .	32
6.16.2.6 uRightBorder . . . . .	32
6.17 emf_settings_t Struct Reference . . . . .	32
6.17.1 Detailed Description . . . . .	33
6.17.2 Field Documentation . . . . .	33
6.17.2.1 BackEMFFlags . . . . .	33
6.17.2.2 Km . . . . .	33
6.17.2.3 L . . . . .	33
6.17.2.4 R . . . . .	33
6.18 encoder_information_t Struct Reference . . . . .	33
6.18.1 Detailed Description . . . . .	34
6.18.2 Field Documentation . . . . .	34
6.18.2.1 Manufacturer . . . . .	34
6.18.2.2 PartNumber . . . . .	34
6.19 encoder_settings_t Struct Reference . . . . .	34

6.19.1 Detailed Description . . . . .	35
6.19.2 Field Documentation . . . . .	35
6.19.2.1 EncoderSettings . . . . .	35
6.19.2.2 MaxCurrentConsumption . . . . .	35
6.19.2.3 MaxOperatingFrequency . . . . .	35
6.19.2.4 SupplyVoltageMax . . . . .	35
6.19.2.5 SupplyVoltageMin . . . . .	35
6.20 engine_advansed_setup_t Struct Reference . . . . .	35
6.20.1 Detailed Description . . . . .	36
6.20.2 Field Documentation . . . . .	36
6.20.2.1 stepcloseloop_Kp_high . . . . .	36
6.20.2.2 stepcloseloop_Kp_low . . . . .	36
6.20.2.3 stepcloseloop_Kw . . . . .	36
6.21 engine_settings_calb_t Struct Reference . . . . .	36
6.21.1 Detailed Description . . . . .	37
6.21.2 Field Documentation . . . . .	37
6.21.2.1 Antiplay . . . . .	37
6.21.2.2 EngineFlags . . . . .	37
6.21.2.3 MicrostepMode . . . . .	37
6.21.2.4 NomCurrent . . . . .	37
6.21.2.5 NomSpeed . . . . .	37
6.21.2.6 NomVoltage . . . . .	37
6.21.2.7 StepsPerRev . . . . .	37
6.22 engine_settings_t Struct Reference . . . . .	38
6.22.1 Detailed Description . . . . .	38
6.22.2 Field Documentation . . . . .	38
6.22.2.1 Antiplay . . . . .	38
6.22.2.2 EngineFlags . . . . .	38
6.22.2.3 MicrostepMode . . . . .	39
6.22.2.4 NomCurrent . . . . .	39
6.22.2.5 NomSpeed . . . . .	39
6.22.2.6 NomVoltage . . . . .	39
6.22.2.7 StepsPerRev . . . . .	39
6.22.2.8 uNomSpeed . . . . .	39
6.23 entype_settings_t Struct Reference . . . . .	39
6.23.1 Detailed Description . . . . .	40
6.23.2 Field Documentation . . . . .	40
6.23.2.1 DriverType . . . . .	40
6.23.2.2 EngineType . . . . .	40

6.24 extended_settings_t Struct Reference . . . . .	40
6.24.1 Detailed Description . . . . .	40
6.25 extio_settings_t Struct Reference . . . . .	40
6.25.1 Detailed Description . . . . .	41
6.25.2 Field Documentation . . . . .	41
6.25.2.1 EXTIOModeFlags . . . . .	41
6.25.2.2 EXTIOSetupFlags . . . . .	41
6.26 feedback_settings_t Struct Reference . . . . .	41
6.26.1 Detailed Description . . . . .	41
6.26.2 Field Documentation . . . . .	42
6.26.2.1 CountsPerTurn . . . . .	42
6.26.2.2 FeedbackFlags . . . . .	42
6.26.2.3 FeedbackType . . . . .	42
6.26.2.4 IPS . . . . .	42
6.27 gear_information_t Struct Reference . . . . .	42
6.27.1 Detailed Description . . . . .	42
6.27.2 Field Documentation . . . . .	43
6.27.2.1 Manufacturer . . . . .	43
6.27.2.2 PartNumber . . . . .	43
6.28 gear_settings_t Struct Reference . . . . .	43
6.28.1 Detailed Description . . . . .	43
6.28.2 Field Documentation . . . . .	43
6.28.2.1 Efficiency . . . . .	43
6.28.2.2 InputInertia . . . . .	44
6.28.2.3 MaxOutputBacklash . . . . .	44
6.28.2.4 RatedInputSpeed . . . . .	44
6.28.2.5 RatedInputTorque . . . . .	44
6.28.2.6 ReductionIn . . . . .	44
6.28.2.7 ReductionOut . . . . .	44
6.29 get_position_calb_t Struct Reference . . . . .	44
6.29.1 Detailed Description . . . . .	44
6.29.2 Field Documentation . . . . .	45
6.29.2.1 EncPosition . . . . .	45
6.29.2.2 Position . . . . .	45
6.30 get_position_t Struct Reference . . . . .	45
6.30.1 Detailed Description . . . . .	45
6.30.2 Field Documentation . . . . .	45
6.30.2.1 EncPosition . . . . .	45
6.30.2.2 uPosition . . . . .	45

6.31	<a href="#">globally_unique_identifier_t Struct Reference</a>	46
6.31.1	<a href="#">Detailed Description</a>	46
6.31.2	<a href="#">Field Documentation</a>	46
6.31.2.1	<a href="#">UniqueID0</a>	46
6.31.2.2	<a href="#">UniqueID1</a>	46
6.31.2.3	<a href="#">UniqueID2</a>	46
6.31.2.4	<a href="#">UniqueID3</a>	46
6.32	<a href="#">hallsensor_information_t Struct Reference</a>	46
6.32.1	<a href="#">Detailed Description</a>	47
6.32.2	<a href="#">Field Documentation</a>	47
6.32.2.1	<a href="#">Manufacturer</a>	47
6.32.2.2	<a href="#">PartNumber</a>	47
6.33	<a href="#">hallsensor_settings_t Struct Reference</a>	47
6.33.1	<a href="#">Detailed Description</a>	48
6.33.2	<a href="#">Field Documentation</a>	48
6.33.2.1	<a href="#">MaxCurrentConsumption</a>	48
6.33.2.2	<a href="#">MaxOperatingFrequency</a>	48
6.33.2.3	<a href="#">SupplyVoltageMax</a>	48
6.33.2.4	<a href="#">SupplyVoltageMin</a>	48
6.34	<a href="#">home_settings_calb_t Struct Reference</a>	48
6.34.1	<a href="#">Detailed Description</a>	49
6.34.2	<a href="#">Field Documentation</a>	49
6.34.2.1	<a href="#">FastHome</a>	49
6.34.2.2	<a href="#">HomeDelta</a>	49
6.34.2.3	<a href="#">HomeFlags</a>	49
6.34.2.4	<a href="#">SlowHome</a>	49
6.35	<a href="#">home_settings_t Struct Reference</a>	49
6.35.1	<a href="#">Detailed Description</a>	50
6.35.2	<a href="#">Field Documentation</a>	50
6.35.2.1	<a href="#">FastHome</a>	50
6.35.2.2	<a href="#">HomeDelta</a>	50
6.35.2.3	<a href="#">HomeFlags</a>	50
6.35.2.4	<a href="#">SlowHome</a>	50
6.35.2.5	<a href="#">uFastHome</a>	50
6.35.2.6	<a href="#">uHomeDelta</a>	50
6.35.2.7	<a href="#">uSlowHome</a>	50
6.36	<a href="#">init_random_t Struct Reference</a>	51
6.36.1	<a href="#">Detailed Description</a>	51
6.36.2	<a href="#">Field Documentation</a>	51

6.36.2.1 key . . . . .	51
6.37 joystick_settings_t Struct Reference . . . . .	51
6.37.1 Detailed Description . . . . .	51
6.37.2 Field Documentation . . . . .	52
6.37.2.1 DeadZone . . . . .	52
6.37.2.2 ExpFactor . . . . .	52
6.37.2.3 JoyCenter . . . . .	52
6.37.2.4 JoyFlags . . . . .	52
6.37.2.5 JoyHighEnd . . . . .	52
6.37.2.6 JoyLowEnd . . . . .	52
6.38 measurements_t Struct Reference . . . . .	52
6.38.1 Detailed Description . . . . .	53
6.38.2 Field Documentation . . . . .	53
6.38.2.1 Error . . . . .	53
6.38.2.2 Length . . . . .	53
6.38.2.3 Speed . . . . .	53
6.39 motor_information_t Struct Reference . . . . .	53
6.39.1 Detailed Description . . . . .	53
6.39.2 Field Documentation . . . . .	54
6.39.2.1 Manufacturer . . . . .	54
6.39.2.2 PartNumber . . . . .	54
6.40 motor_settings_t Struct Reference . . . . .	54
6.40.1 Detailed Description . . . . .	55
6.40.2 Field Documentation . . . . .	55
6.40.2.1 DetentTorque . . . . .	55
6.40.2.2 MaxCurrent . . . . .	55
6.40.2.3 MaxCurrentTime . . . . .	56
6.40.2.4 MaxSpeed . . . . .	56
6.40.2.5 MechanicalTimeConstant . . . . .	56
6.40.2.6 MotorType . . . . .	56
6.40.2.7 NoLoadCurrent . . . . .	56
6.40.2.8 NoLoadSpeed . . . . .	56
6.40.2.9 NominalCurrent . . . . .	56
6.40.2.10NominalPower . . . . .	56
6.40.2.11NominalSpeed . . . . .	56
6.40.2.12NominalTorque . . . . .	57
6.40.2.13NominalVoltage . . . . .	57
6.40.2.14Phases . . . . .	57
6.40.2.15Poles . . . . .	57

6.40.2.16 RotorInertia	57
6.40.2.17 SpeedConstant	57
6.40.2.18 SpeedTorqueGradient	57
6.40.2.19 StallTorque	57
6.40.2.20 TorqueConstant	57
6.40.2.21 WindingInductance	58
6.40.2.22 WindingResistance	58
6.41 move_settings_calb_t Struct Reference	58
6.41.1 Detailed Description	58
6.41.2 Field Documentation	58
6.41.2.1 Accel	58
6.41.2.2 AntiplaySpeed	58
6.41.2.3 Decel	59
6.41.2.4 MoveFlags	59
6.41.2.5 Speed	59
6.42 move_settings_t Struct Reference	59
6.42.1 Detailed Description	59
6.42.2 Field Documentation	59
6.42.2.1 Accel	59
6.42.2.2 AntiplaySpeed	60
6.42.2.3 Decel	60
6.42.2.4 MoveFlags	60
6.42.2.5 Speed	60
6.42.2.6 uAntiplaySpeed	60
6.42.2.7 uSpeed	60
6.43 nonvolatile_memory_t Struct Reference	60
6.43.1 Detailed Description	60
6.43.2 Field Documentation	61
6.43.2.1 UserData	61
6.44 pid_settings_t Struct Reference	61
6.44.1 Detailed Description	61
6.45 power_settings_t Struct Reference	61
6.45.1 Detailed Description	62
6.45.2 Field Documentation	62
6.45.2.1 CurrentSetTime	62
6.45.2.2 CurrReductDelay	62
6.45.2.3 HoldCurrent	62
6.45.2.4 PowerFlags	62
6.45.2.5 PowerOffDelay	62

6.46 secure_settings_t Struct Reference . . . . .	62
6.46.1 Detailed Description . . . . .	63
6.46.2 Field Documentation . . . . .	63
6.46.2.1 Criticallpwr . . . . .	63
6.46.2.2 Criticallusb . . . . .	63
6.46.2.3 CriticalT . . . . .	63
6.46.2.4 CriticalUpwr . . . . .	63
6.46.2.5 CriticalUusb . . . . .	64
6.46.2.6 Flags . . . . .	64
6.46.2.7 LowUpwrOff . . . . .	64
6.46.2.8 MinimumUusb . . . . .	64
6.47 serial_number_t Struct Reference . . . . .	64
6.47.1 Detailed Description . . . . .	64
6.47.2 Field Documentation . . . . .	64
6.47.2.1 Key . . . . .	64
6.47.2.2 Major . . . . .	65
6.47.2.3 Minor . . . . .	65
6.47.2.4 Release . . . . .	65
6.47.2.5 SN . . . . .	65
6.48 set_position_calb_t Struct Reference . . . . .	65
6.48.1 Detailed Description . . . . .	65
6.48.2 Field Documentation . . . . .	65
6.48.2.1 EncPosition . . . . .	65
6.48.2.2 PosFlags . . . . .	65
6.48.2.3 Position . . . . .	66
6.49 set_position_t Struct Reference . . . . .	66
6.49.1 Detailed Description . . . . .	66
6.49.2 Field Documentation . . . . .	66
6.49.2.1 EncPosition . . . . .	66
6.49.2.2 PosFlags . . . . .	66
6.49.2.3 uPosition . . . . .	66
6.50 stage_information_t Struct Reference . . . . .	66
6.50.1 Detailed Description . . . . .	67
6.50.2 Field Documentation . . . . .	67
6.50.2.1 Manufacturer . . . . .	67
6.50.2.2 PartNumber . . . . .	67
6.51 stage_name_t Struct Reference . . . . .	67
6.51.1 Detailed Description . . . . .	67
6.51.2 Field Documentation . . . . .	68

6.51.2.1 PositionerName . . . . .	68
6.52 stage_settings_t Struct Reference . . . . .	68
6.52.1 Detailed Description . . . . .	68
6.52.2 Field Documentation . . . . .	68
6.52.2.1 HorizontalLoadCapacity . . . . .	68
6.52.2.2 LeadScrewPitch . . . . .	69
6.52.2.3 MaxCurrentConsumption . . . . .	69
6.52.2.4 MaxSpeed . . . . .	69
6.52.2.5 SupplyVoltageMax . . . . .	69
6.52.2.6 SupplyVoltageMin . . . . .	69
6.52.2.7 TravelRange . . . . .	69
6.52.2.8 Units . . . . .	69
6.52.2.9 VerticalLoadCapacity . . . . .	69
6.53 status_calb_t Struct Reference . . . . .	69
6.53.1 Detailed Description . . . . .	70
6.53.2 Field Documentation . . . . .	70
6.53.2.1 CmdBufFreeSpace . . . . .	70
6.53.2.2 CurPosition . . . . .	71
6.53.2.3 CurSpeed . . . . .	71
6.53.2.4 CurT . . . . .	71
6.53.2.5 EncPosition . . . . .	71
6.53.2.6 EncSts . . . . .	71
6.53.2.7 Flags . . . . .	71
6.53.2.8 GPIOFlags . . . . .	71
6.53.2.9 Ipwr . . . . .	71
6.53.2.10Iusb . . . . .	71
6.53.2.11MoveSts . . . . .	71
6.53.2.12MvCmdSts . . . . .	71
6.53.2.13PWRSts . . . . .	71
6.53.2.14Upwr . . . . .	72
6.53.2.15Uusb . . . . .	72
6.53.2.16WindSts . . . . .	72
6.54 status_t Struct Reference . . . . .	72
6.54.1 Detailed Description . . . . .	73
6.54.2 Field Documentation . . . . .	73
6.54.2.1 CmdBufFreeSpace . . . . .	73
6.54.2.2 CurPosition . . . . .	73
6.54.2.3 CurSpeed . . . . .	73
6.54.2.4 CurT . . . . .	73

6.54.2.5	EncPosition	73
6.54.2.6	EncSts	73
6.54.2.7	Flags	73
6.54.2.8	GPIOFlags	74
6.54.2.9	Ipwr	74
6.54.2.10	Iusb	74
6.54.2.11	MoveSts	74
6.54.2.12	MvCmdSts	74
6.54.2.13	PWRSts	74
6.54.2.14	uCurPosition	74
6.54.2.15	uCurSpeed	74
6.54.2.16	Upwr	74
6.54.2.17	Uusb	74
6.54.2.18	WindSts	74
6.55	sync_in_settings_calb_t Struct Reference	75
6.55.1	Detailed Description	75
6.55.2	Field Documentation	75
6.55.2.1	ClutterTime	75
6.55.2.2	Position	75
6.55.2.3	Speed	75
6.55.2.4	SyncInFlags	75
6.56	sync_in_settings_t Struct Reference	75
6.56.1	Detailed Description	76
6.56.2	Field Documentation	76
6.56.2.1	ClutterTime	76
6.56.2.2	Speed	76
6.56.2.3	SyncInFlags	76
6.56.2.4	uPosition	76
6.56.2.5	uSpeed	77
6.57	sync_out_settings_calb_t Struct Reference	77
6.57.1	Detailed Description	77
6.57.2	Field Documentation	77
6.57.2.1	Accuracy	77
6.57.2.2	SyncOutFlags	77
6.57.2.3	SyncOutPeriod	77
6.57.2.4	SyncOutPulseSteps	78
6.58	sync_out_settings_t Struct Reference	78
6.58.1	Detailed Description	78
6.58.2	Field Documentation	78

6.58.2.1 Accuracy . . . . .	78
6.58.2.2 SyncOutFlags . . . . .	78
6.58.2.3 SyncOutPeriod . . . . .	79
6.58.2.4 SyncOutPulseSteps . . . . .	79
6.58.2.5 uAccuracy . . . . .	79
6.59 uart_settings_t Struct Reference . . . . .	79
6.59.1 Detailed Description . . . . .	79
6.59.2 Field Documentation . . . . .	79
6.59.2.1 UARTSetupFlags . . . . .	79
<b>7 File Documentation</b>	<b>80</b>
7.1 ximc.h File Reference . . . . .	80
7.1.1 Detailed Description . . . . .	104
7.1.2 Macro Definition Documentation . . . . .	104
7.1.2.1 ALARM_ON_DRIVER_OVERHEATING . . . . .	104
7.1.2.2 BACK_EMF_INDUCTANCE_AUTO . . . . .	104
7.1.2.3 BACK_EMF_KM_AUTO . . . . .	104
7.1.2.4 BACK_EMF_RESISTANCE_AUTO . . . . .	104
7.1.2.5 BORDER_IS_ENCODER . . . . .	105
7.1.2.6 BORDER_STOP_LEFT . . . . .	105
7.1.2.7 BORDER_STOP_RIGHT . . . . .	105
7.1.2.8 BORDERS_SWAP_MISSET_DETECTION . . . . .	105
7.1.2.9 BRAKE_ENABLED . . . . .	105
7.1.2.10 BRAKE_ENG_PWROFF . . . . .	105
7.1.2.11 CONTROL_BTN_LEFT_PUSHED_OPEN . . . . .	105
7.1.2.12 CONTROL_BTN_RIGHT_PUSHED_OPEN . . . . .	105
7.1.2.13 CONTROL_MODE_BITS . . . . .	105
7.1.2.14 CONTROL_MODE_JOY . . . . .	105
7.1.2.15 CONTROL_MODE_LR . . . . .	105
7.1.2.16 CONTROL_MODE_OFF . . . . .	105
7.1.2.17 CTP_ALARM_ON_ERROR . . . . .	106
7.1.2.18 CTP_BASE . . . . .	106
7.1.2.19 CTP_ENABLED . . . . .	106
7.1.2.20 CTP_ERROR_CORRECTION . . . . .	106
7.1.2.21 DRIVER_TYPE_DISCRETE_FET . . . . .	106
7.1.2.22 DRIVER_TYPE_EXTERNAL . . . . .	106
7.1.2.23 DRIVER_TYPE_INTEGRATE . . . . .	106
7.1.2.24 EEPROM_PRECEDENCE . . . . .	106
7.1.2.25 ENC_STATE_ABSENT . . . . .	106

7.1.2.26	ENC_STATE_MALFUNC	106
7.1.2.27	ENC_STATE_OK	106
7.1.2.28	ENC_STATE_REVERS	107
7.1.2.29	ENC_STATE_UNKNOWN	107
7.1.2.30	ENDER_SW1_ACTIVE_LOW	107
7.1.2.31	ENDER_SW2_ACTIVE_LOW	107
7.1.2.32	ENDER_SWAP	107
7.1.2.33	ENGINE_ACCEL_ON	107
7.1.2.34	ENGINE_ANTIPLAY	107
7.1.2.35	ENGINE_CURRENT_AS_RMS	107
7.1.2.36	ENGINE_LIMIT_CURR	107
7.1.2.37	ENGINE_LIMIT_RPM	107
7.1.2.38	ENGINE_LIMIT_VOLT	108
7.1.2.39	ENGINE_MAX_SPEED	108
7.1.2.40	ENGINE_REVERSE	108
7.1.2.41	ENGINE_TYPE_2DC	108
7.1.2.42	ENGINE_TYPE_BRUSHLESS	108
7.1.2.43	ENGINE_TYPE_DC	108
7.1.2.44	ENGINE_TYPE_NONE	108
7.1.2.45	ENGINE_TYPE_STEP	108
7.1.2.46	ENGINE_TYPE_TEST	108
7.1.2.47	ENUMERATE_PROBE	108
7.1.2.48	EXTIO_SETUP_INVERT	109
7.1.2.49	EXTIO_SETUP_MODE_IN_ALARM	109
7.1.2.50	EXTIO_SETUP_MODE_IN_BITS	109
7.1.2.51	EXTIO_SETUP_MODE_IN_HOME	109
7.1.2.52	EXTIO_SETUP_MODE_IN_MOVR	109
7.1.2.53	EXTIO_SETUP_MODE_IN_NOP	109
7.1.2.54	EXTIO_SETUP_MODE_IN_PWOF	109
7.1.2.55	EXTIO_SETUP_MODE_IN_STOP	109
7.1.2.56	EXTIO_SETUP_MODE_OUT_ALARM	109
7.1.2.57	EXTIO_SETUP_MODE_OUT_BITS	109
7.1.2.58	EXTIO_SETUP_MODE_OUT_MOTOR_ON	109
7.1.2.59	EXTIO_SETUP_MODE_OUT_MOVING	109
7.1.2.60	EXTIO_SETUP_MODE_OUT_OFF	110
7.1.2.61	EXTIO_SETUP_MODE_OUT_ON	110
7.1.2.62	EXTIO_SETUP_OUTPUT	110
7.1.2.63	FEEDBACK_EMF	110
7.1.2.64	FEEDBACK_ENC_REVERSE	110

7.1.2.65	FEEDBACK_ENC_TYPE_AUTO	110
7.1.2.66	FEEDBACK_ENC_TYPE_BITS	110
7.1.2.67	FEEDBACK_ENC_TYPE_DIFFERENTIAL	110
7.1.2.68	FEEDBACK_ENC_TYPE_SINGLE_ENDED	110
7.1.2.69	FEEDBACK_ENCODER	110
7.1.2.70	FEEDBACK_ENCODER_MEDIATED	110
7.1.2.71	FEEDBACK_NONE	110
7.1.2.72	H_BRIDGE_ALERT	111
7.1.2.73	HOME_DIR_FIRST	111
7.1.2.74	HOME_DIR_SECOND	111
7.1.2.75	HOME_HALF_MV	111
7.1.2.76	HOME_MV_SEC_EN	111
7.1.2.77	HOME_STOP_FIRST_BITS	111
7.1.2.78	HOME_STOP_FIRST_LIM	111
7.1.2.79	HOME_STOP_FIRST_REV	111
7.1.2.80	HOME_STOP_FIRST_SYN	111
7.1.2.81	HOME_STOP_SECOND_BITS	111
7.1.2.82	HOME_STOP_SECOND_LIM	111
7.1.2.83	HOME_STOP_SECOND_REV	112
7.1.2.84	HOME_STOP_SECOND_SYN	112
7.1.2.85	HOME_USE_FAST	112
7.1.2.86	JOY_REVERSE	112
7.1.2.87	LOW_UPWR_PROTECTION	112
7.1.2.88	MICROSTEP_MODE_FRAC_128	112
7.1.2.89	MICROSTEP_MODE_FRAC_16	112
7.1.2.90	MICROSTEP_MODE_FRAC_2	112
7.1.2.91	MICROSTEP_MODE_FRAC_256	112
7.1.2.92	MICROSTEP_MODE_FRAC_32	112
7.1.2.93	MICROSTEP_MODE_FRAC_4	112
7.1.2.94	MICROSTEP_MODE_FRAC_64	112
7.1.2.95	MICROSTEP_MODE_FRAC_8	113
7.1.2.96	MICROSTEP_MODE_FULL	113
7.1.2.97	MOVE_STATE_ANTIPLAY	113
7.1.2.98	MOVE_STATE_MOVING	113
7.1.2.99	MOVE_STATE_TARGET_SPEED	113
7.1.2.100	MVCMD_ERROR	113
7.1.2.101	MVCMD_HOME	113
7.1.2.102	MVCMD_LEFT	113
7.1.2.103	MVCMD_LOFT	113

7.1.2.104MVCMD_MOVE	113
7.1.2.105MVCMD_MOVR	113
7.1.2.106MVCMD_NAME_BITS	114
7.1.2.107MVCMD_RIGHT	114
7.1.2.108MVCMD_RUNNING	114
7.1.2.109MVCMD_SSTP	114
7.1.2.110MVCMD_STOP	114
7.1.2.111MVCMD_UKNWN	114
7.1.2.112POWER_OFF_ENABLED	114
7.1.2.113POWER_REDUCT_ENABLED	114
7.1.2.114POWER_SMOOTH_CURRENT	114
7.1.2.115PWR_STATE_MAX	114
7.1.2.116PWR_STATE_NORM	114
7.1.2.117PWR_STATE_OFF	114
7.1.2.118PWR_STATE_REDUCT	115
7.1.2.119PWR_STATE_UNKNOWN	115
7.1.2.120REV_SENS_INV	115
7.1.2.121RPM_DIV_1000	115
7.1.2.122SETPOS_IGNORE_ENCODER	115
7.1.2.123SETPOS_IGNORE_POSITION	115
7.1.2.124STATE_ALARM	115
7.1.2.125STATE_BORDERS_SWAP_MISSET	115
7.1.2.126STATE_BRAKE	115
7.1.2.127STATE_BUTTON_LEFT	115
7.1.2.128STATE_BUTTON_RIGHT	115
7.1.2.129STATE_CONTR	116
7.1.2.130STATE_CONTROLLER_OVERHEAT	116
7.1.2.131STATE_CTP_ERROR	116
7.1.2.132STATE_DIG_SIGNAL	116
7.1.2.133STATE_EEPROM_CONNECTED	116
7.1.2.134STATE_ENC_A	116
7.1.2.135STATE_ENC_B	116
7.1.2.136STATE_ENGINE_RESPONSE_ERROR	116
7.1.2.137STATE_ERRC	116
7.1.2.138STATE_ERRD	116
7.1.2.139STATE_ERRV	116
7.1.2.140STATE_EXTIO_ALARM	116
7.1.2.141STATE_GPIO_LEVEL	117
7.1.2.142STATE_GPIO_PINOUT	117

7.1.2.143	STATE_LEFT_EDGE	117
7.1.2.144	STATE_LOW_USB_VOLTAGE	117
7.1.2.145	STATE_OVERLOAD_POWER_CURRENT	117
7.1.2.146	STATE_OVERLOAD_POWER_VOLTAGE	117
7.1.2.147	STATE_OVERLOAD_USB_CURRENT	117
7.1.2.148	STATE_OVERLOAD_USB_VOLTAGE	117
7.1.2.149	STATE_POWER_OVERHEAT	117
7.1.2.150	STATE_REV_SENSOR	117
7.1.2.151	STATE_RIGHT_EDGE	117
7.1.2.152	STATE_SECUR	117
7.1.2.153	STATE_SYNC_INPUT	118
7.1.2.154	STATE_SYNC_OUTPUT	118
7.1.2.155	SYNCIN_ENABLED	118
7.1.2.156	SYNCIN_GOTOPOSITION	118
7.1.2.157	SYNCIN_INVERT	118
7.1.2.158	SYNCOUT_ENABLED	118
7.1.2.159	SYNCOUT_IN_STEPS	118
7.1.2.160	SYNCOUT_INVERT	118
7.1.2.161	SYNCOUT_ONPERIOD	118
7.1.2.162	SYNCOUT_ONSTART	118
7.1.2.163	SYNCOUT_ONSTOP	118
7.1.2.164	SYNCOUT_STATE	119
7.1.2.165	UART_PARITY_BITS	119
7.1.2.166	WIND_A.STATE_ABSENT	119
7.1.2.167	WIND_A.STATE_MALFUNC	119
7.1.2.168	WIND_A.STATE_OK	119
7.1.2.169	WIND_A.STATE_UNKNOWN	119
7.1.2.170	WIND_B.STATE_ABSENT	119
7.1.2.171	WIND_B.STATE_MALFUNC	119
7.1.2.172	WIND_B.STATE_OK	119
7.1.2.173	WIND_B.STATE_UNKNOWN	119
7.1.2.174	XIMC_API	119
7.1.3	Typedef Documentation	120
7.1.3.1	logging_callback_t	120
7.1.4	Function Documentation	120
7.1.4.1	close_device	120
7.1.4.2	command_clear_fram	120
7.1.4.3	command_eeread_settings	120
7.1.4.4	command_eesave_settings	120

7.1.4.5	<a href="#">command_home</a>	121
7.1.4.6	<a href="#">command_homezero</a>	121
7.1.4.7	<a href="#">command_left</a>	121
7.1.4.8	<a href="#">command_loft</a>	122
7.1.4.9	<a href="#">command_move</a>	122
7.1.4.10	<a href="#">command_move_calb</a>	122
7.1.4.11	<a href="#">command_movr</a>	122
7.1.4.12	<a href="#">command_movr_calb</a>	123
7.1.4.13	<a href="#">command_power_off</a>	123
7.1.4.14	<a href="#">command_read_robust_settings</a>	123
7.1.4.15	<a href="#">command_read_settings</a>	124
7.1.4.16	<a href="#">command_reset</a>	124
7.1.4.17	<a href="#">command_right</a>	124
7.1.4.18	<a href="#">command_save_robust_settings</a>	124
7.1.4.19	<a href="#">command_save_settings</a>	124
7.1.4.20	<a href="#">command_sstp</a>	124
7.1.4.21	<a href="#">command_start_measurements</a>	125
7.1.4.22	<a href="#">command_stop</a>	125
7.1.4.23	<a href="#">command_update_firmware</a>	125
7.1.4.24	<a href="#">command_wait_for_stop</a>	125
7.1.4.25	<a href="#">command_zero</a>	126
7.1.4.26	<a href="#">enumerate_devices</a>	126
7.1.4.27	<a href="#">free_enumerate_devices</a>	126
7.1.4.28	<a href="#">get_accessories_settings</a>	126
7.1.4.29	<a href="#">get_analog_data</a>	127
7.1.4.30	<a href="#">get_bootloader_version</a>	127
7.1.4.31	<a href="#">get_brake_settings</a>	127
7.1.4.32	<a href="#">get_calibration_settings</a>	127
7.1.4.33	<a href="#">get_chart_data</a>	127
7.1.4.34	<a href="#">get_control_settings</a>	128
7.1.4.35	<a href="#">get_control_settings_calb</a>	128
7.1.4.36	<a href="#">get_controller_name</a>	128
7.1.4.37	<a href="#">get_ctp_settings</a>	129
7.1.4.38	<a href="#">get_debug_read</a>	129
7.1.4.39	<a href="#">get_device_count</a>	129
7.1.4.40	<a href="#">get_device_information</a>	129
7.1.4.41	<a href="#">get_device_name</a>	130
7.1.4.42	<a href="#">get_edges_settings</a>	130
7.1.4.43	<a href="#">get_edges_settings_calb</a>	130

7.1.4.44	<a href="#">get_emf_settings</a>	131
7.1.4.45	<a href="#">get_encoder_information</a>	131
7.1.4.46	<a href="#">get_encoder_settings</a>	131
7.1.4.47	<a href="#">get_engine_advanced_setup</a>	131
7.1.4.48	<a href="#">get_engine_settings</a>	132
7.1.4.49	<a href="#">get_engine_settings_calb</a>	132
7.1.4.50	<a href="#">get_entype_settings</a>	132
7.1.4.51	<a href="#">get_enumerate_device_controller_name</a>	132
7.1.4.52	<a href="#">get_enumerate_device_information</a>	133
7.1.4.53	<a href="#">get_enumerate_device_network_information</a>	133
7.1.4.54	<a href="#">get_enumerate_device_serial</a>	133
7.1.4.55	<a href="#">get_enumerate_device_stage_name</a>	134
7.1.4.56	<a href="#">get_extended_settings</a>	134
7.1.4.57	<a href="#">get_extio_settings</a>	134
7.1.4.58	<a href="#">get_feedback_settings</a>	134
7.1.4.59	<a href="#">get_firmware_version</a>	135
7.1.4.60	<a href="#">get_gear_information</a>	135
7.1.4.61	<a href="#">get_gear_settings</a>	135
7.1.4.62	<a href="#">get_globally_unique_identifier</a>	135
7.1.4.63	<a href="#">get_hallsensor_information</a>	136
7.1.4.64	<a href="#">get_hallsensor_settings</a>	136
7.1.4.65	<a href="#">get_home_settings</a>	136
7.1.4.66	<a href="#">get_home_settings_calb</a>	136
7.1.4.67	<a href="#">get_init_random</a>	137
7.1.4.68	<a href="#">get_joystick_settings</a>	137
7.1.4.69	<a href="#">get_measurements</a>	137
7.1.4.70	<a href="#">get_motor_information</a>	138
7.1.4.71	<a href="#">get_motor_settings</a>	138
7.1.4.72	<a href="#">get_move_settings</a>	138
7.1.4.73	<a href="#">get_move_settings_calb</a>	138
7.1.4.74	<a href="#">get_nonvolatile_memory</a>	138
7.1.4.75	<a href="#">get_pid_settings</a>	139
7.1.4.76	<a href="#">get_position</a>	139
7.1.4.77	<a href="#">get_position_calb</a>	139
7.1.4.78	<a href="#">get_power_settings</a>	139
7.1.4.79	<a href="#">get_secure_settings</a>	140
7.1.4.80	<a href="#">get_serial_number</a>	140
7.1.4.81	<a href="#">get_stage_information</a>	140
7.1.4.82	<a href="#">get_stage_name</a>	140

7.1.4.83	get_stage_settings	140
7.1.4.84	get_status	141
7.1.4.85	get_status_calb	141
7.1.4.86	get_sync_in_settings	141
7.1.4.87	get_sync_in_settings_calb	141
7.1.4.88	get_sync_out_settings	142
7.1.4.89	get_sync_out_settings_calb	142
7.1.4.90	get_uart_settings	142
7.1.4.91	goto_firmware	143
7.1.4.92	has_firmware	143
7.1.4.93	load_correction_table	143
7.1.4.94	logging_callback_stderr_narrow	144
7.1.4.95	logging_callback_stderr_wide	144
7.1.4.96	msec_sleep	144
7.1.4.97	open_device	144
7.1.4.98	probe_device	145
7.1.4.99	reset_locks	145
7.1.4.100	service_command_updf	145
7.1.4.101	set_accessories_settings	145
7.1.4.102	set_bindy_key	146
7.1.4.103	set_brake_settings	146
7.1.4.104	set_calibration_settings	146
7.1.4.105	set_control_settings	146
7.1.4.106	set_control_settings_calb	147
7.1.4.107	set_controller_name	147
7.1.4.108	set_correction_table	147
7.1.4.109	set_ctp_settings	148
7.1.4.110	set_debug_write	148
7.1.4.111	set_edges_settings	148
7.1.4.112	set_edges_settings_calb	148
7.1.4.113	set_emf_settings	149
7.1.4.114	set_encoder_information	149
7.1.4.115	set_encoder_settings	149
7.1.4.116	set_engine_advanced_setup	150
7.1.4.117	set_engine_settings	150
7.1.4.118	set_engine_settings_calb	150
7.1.4.119	set_entype_settings	151
7.1.4.120	set_extended_settings	151
7.1.4.121	set_extio_settings	151

7.1.4.122set_feedback_settings . . . . .	151
7.1.4.123set_gear_information . . . . .	152
7.1.4.124set_gear_settings . . . . .	152
7.1.4.125set_hallsensor_information . . . . .	152
7.1.4.126set_hallsensor_settings . . . . .	152
7.1.4.127set_home_settings . . . . .	153
7.1.4.128set_home_settings_calb . . . . .	153
7.1.4.129set_joystick_settings . . . . .	153
7.1.4.130set_logging_callback . . . . .	154
7.1.4.131set_motor_information . . . . .	154
7.1.4.132set_motor_settings . . . . .	154
7.1.4.133set_move_settings . . . . .	154
7.1.4.134set_move_settings_calb . . . . .	155
7.1.4.135set_nonvolatile_memory . . . . .	155
7.1.4.136set_pid_settings . . . . .	155
7.1.4.137set_position . . . . .	155
7.1.4.138set_position_calb . . . . .	156
7.1.4.139set_power_settings . . . . .	156
7.1.4.140set_secure_settings . . . . .	156
7.1.4.141set_serial_number . . . . .	156
7.1.4.142set_stage_information . . . . .	157
7.1.4.143set_stage_name . . . . .	157
7.1.4.144set_stage_settings . . . . .	157
7.1.4.145set_sync_in_settings . . . . .	157
7.1.4.146set_sync_in_settings_calb . . . . .	158
7.1.4.147set_sync_out_settings . . . . .	158
7.1.4.148set_sync_out_settings_calb . . . . .	158
7.1.4.149set_uart_settings . . . . .	159
7.1.4.150write_key . . . . .	159
7.1.4.151ximc_fix_usbser_sys . . . . .	159
7.1.4.152ximc_version . . . . .	159

# 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](#).

### 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 hardware using ready-made software or using libraries for programming languages: C / C ++, C #, JAVA, Visual Basic, Python 2/3, .NET, Delphi, integration with MS Visual Studio programming environments, gcc, Xcode.
- Works with scientific development environments by integrating LabVIEW and MATLAB;

### 1.2 What can do libximc library

- Libximc manages hardware 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 Mac OS X
- 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 (XiLab 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 suggestions, errors and ideas. We appreciate your suggestions and try to make our product better. Please post your questions [here](#). Your ideas and comments send a e-mail: [8smc4@standa.lt](mailto:8smc4@standa.lt)

# Chapter 2

## Introduction

### 2.1 About library

This document contains all information about libximc library. It utilizes well known virtual COM-port interface, so you can use it on Windows 7, Windows, Vista, Windows XP, Windows Server 2003, Windows 2000, Linux, Mac OS X. 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.

### 2.2 System requirements

#### 2.2.1 For rebuilding library

On Windows:

- Windows 2000 or later, 64-bit system (if compiling both architectures) or 32-bit system.
- Microsoft Visual C++ 2013 or later
- cygwin with tar, bison, flex, curl installed
- 7z

On Linux:

- 64-bit or/and 32-bit system system
- gcc 4 or later
- common autotools: autoconf, autoheader, aclocal, automake, autoreconf, libtool
- gmake
- doxygen - for building docs
- LaTeX distribution (teTeX or texlive) - for building docs
- flex 2.5.30+
- bison 2.3+
- mercurial (for building developer version from hg)

On Mac OS X:

- XCode 4
- doxygen
- mactex
- autotools
- mercurial (for building developer version from hg)

If mercurial is used, please enable 'purge' extension by adding to `~/.hgrc` following lines:

```
[extensions]
hgext.purge=
```

### 2.2.2 For using library

Supported operating systems (32 or 64 bit) and environment requirements:

- Mac OS X 10.6
- Windows 2000 or later
- Autotools-compatible unix. Package is installed from sources.
- Linux debian-based 32 and 64 bit. DEB package is built against Debian Squeeze 7
- Linux debian-based ARM. DEB package is built on Ubuntu 14.04
- Linux rpm-based. RPM is built against OpenSUSE 12
- Java 7 64-bit or 32-bit
- .NET 2.0 (32-bit only)
- Delphi (32-bit only)

Build requirements:

- Windows: Microsoft Visual C++ 2013 or mingw (currently not supported)
- UNIX: gcc 4, gmake
- Mac OS X: XCode 4
- JDK 7

## Chapter 3

# How to rebuild library

### 3.1 Building on generic UNIX

Generic version could be built with standard autotools.

```
./build.sh lib
```

Built files (library, headers, documentation) are installed to `./dist/local` directory. It is a generic developer build. Sometimes you need to specify additional parameters to command line for your machine. Please look to following OS sections.

### 3.2 Building on debian-based linux systems

Requirement: 64-bit and 32-bit debian system, ubuntu Typical set of packages: gcc, autotools, autoconf, libtool, dpkg-dev, flex, libfl-dev, bison, doxygen, texlive, mercurial Full set of packages: apt-get install ruby1.9.1 debhelper vim sudo g++ mercurial git curl make cmake autotools-dev automake autoconf libtool default-jre-headless default-jdk openjdk-6-jdk dpkg-dev lintian texlive texlive-latex-extra texlive-lang-cyrillic dh-autoreconf hardening-wrapper bison flex libfl-dev doxygen lsb-release pkg-config check For ARM cross-compiling install gcc-arm-linux-gnueabi from your ARM toolchain.

It's required to match library and host architecture: 64-bit library can be built only at 64-bit host, 32-bit library - only at 32-bit host. ARM library is built with armhf cross-compiler gcc-arm-linux-gnueabi.

To build library and package invoke a script:

```
$ ./build.sh libdeb
```

For ARM library replace 'libdeb' with 'libdebarm'.

Grab packages from `./ximc/deb` and locally installed binaries from `./dist/local`.

### 3.3 Building on redhat-based linux systems

Requirement: 64-bit redhat-based system (Fedora, Red Hat, SUSE) Typical set of packages: gcc, autotools, autoconf, libtool, flex, libfl-dev, bison, doxygen, texlive, mercurial Full set of packages: autoconf automake bison doxygen flex libfl-dev gcc gcc-32bit gcc-c++ gcc-c++-32bit java-1.7.0-openjdk java-1.7.0-openjdk-devel libtool lsb-release make mercurial rpm-build rpm-devel rpmlint texlive texlive-fonts-extra texlive-latex

It's possible to build both 32- and 64-bit libraries on 64-bit host system. 64-bit library can't be built on 32-bit system.

To build library and package invoke a script:

```
$ ./build.sh librpm
```

Grab packages from `./ximc/rpm` and locally installed binaries from `./dist/local`.

## 3.4 Buliding on Mac OS X

To build and package a script invoke a script:

```
$ ./build.sh libosx
```

Built library (classical and framework), examples (classical and `.app`), documentation are located at `./ximc/macosex`, locally installed binaries from `./dist/local`.

## 3.5 Buliding on Windows

Requirements: 64-bit windows (build script builds both architectures), cygwin (must be installed to a default path), mercurial.

Invoke a script:

```
$ ./build.bat
```

Grab packages from `./deb/win32` and `./deb/win64`

To build debug version of the library set environment variable "DEBUG" to "true" before running the build script.

## 3.6 Source code access

XIMC source codes are given under special request.

## Chapter 4

# How to use with...

Library usage can be examined from test application testapp. Non-C languages are supported because library supports stdcall calling convention and so can be used with a variety of languages.

C test project is located at 'examples/testapp' directory, C# test project - at 'examples/test\_CSharp', VB.NET - 'examples/test\_VBNET', Delphi 6 - 'examples/test\_Delphi', sample bindings for MATLAB - 'examples/test\_MATLAB', for Java - 'examples/test\_Java', for Python - 'examples/test\_Python'. Development kit also contains precompiled examples: testapp and testappeasy as 32 and 64-bit applications for Windows and 64-bit application for osx, test\_CSharp, test\_VBNET, test\_Delphi - 32-bit only, test\_Java is architecture-independent, test\_MATLAB and test\_Python are runtime-interpreted.

NOTE: SDK requires Microsoft Visual C++ Redistributable Package (provided with SDK - vc\_redist.x86 or vc\_redist.x64)

NOTE: 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).

### 4.1 Usage with C

#### 4.1.1 Visual C++

Testapp can be built using testapp.sln. Library must be compiled with MS Visual C++ too, mingw-library isn't supported. 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).

#### 4.1.2 CodeBlocks

Testapp can be built using test\_CodeBlocks.cbp. Library must be compiled with MS Visual C++ too, mingw-library isn't supported. Make sure that Microsoft Visual C++ Redistributable Package is installed. \*

Open solution examples/test\_CodeBlocks/test\_CodeBlocks.cbp, build and run from the IDE.

#### 4.1.3 MinGW

MinGW is a port of GCC to win32 platform. It's required to install MinGW package. Currently not supported

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 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).

#### 4.1.4 C++ Builder

First of all you should create C++ Builder-style import library. Visual C++ library is not compatible with BCB. Invoke:

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

Then compile test application:

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

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).

#### 4.1.5 XCode

Test app should be built with XCode project testapp.xcodeproj. Library is a Mac OS X 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 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).

#### 4.1.6 GCC

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

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

Copy file /usr/share/libximc/keyfile.sqlite project directory:

```
$ cp /usr/share/libximc/keyfile.sqlite .
```

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 OS X 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 OS X 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).

## 4.2 .NET

Wrapper assembly for libximc.dll is wrappers/csharp/ximcnet.dll. It is provided with two different architectures. Supports the platform .NET from 2.0. to 4.0.

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.

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).

## 4.3 Delphi

Wrapper for libximc.dll is a unit wrappers/delphi/ximc.pas

Console test application for is located at test\_Delphi. Tested with Delphi 6 and only 32-bit version.

Just compile, place DLL near the executable and run program.

In case of the 8SMC4-USB-Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in test\_Delphi.dpr file before build (see enum\_hints variable).

## 4.4 Java

How to run example on Linux. Navigate to ximc-2.x.x./examples/test\_Java/compiled/ and run:

```
$ cp /usr/share/libximc/keyfile.sqlite .
$ java -cp /usr/share/java/libjximc.jar:test_Java.jar ru.ximc.TestJava
```

How to run example on Windows or Mac. Navigate to ximc-2.x.x./examples/test\_Java/compiled/. Copy contents of ximc-2.x.x/ximc/win64 or ximc-2.x.x/ximc/macosx accordingly to the current directory. Then run:

```
$ java -classpath libjximc.jar -classpath test_Java.jar ru.ximc.TestJava
```

How to modify and recompile an example. Navigate 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/libjximc.jar -Xlint ru/ximc/TestJava.java
```

or for windows or mac

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

Then build a jar:

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

In case of the 8SMC4-USB-Eth1 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).

## 4.5 Python

Change current directory to the examples/test\_Python. For correct usage of the library libximc, the example uses the file wrapper, crossplatform\wrappers\python\pyximc.py with a description of the structures of the library.

Before launch:

On OS X: copy library ximc/macosx/libximc.framework to the current directory.

On Linux: you may need to set LD\_LIBRARY\_PATH so Python can locate libraries with RPATH. For example, you may need:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:'pwd'
```

On Windows before the start nothing needs to be done. All necessary communication and dependencies are registered in the example code. Libraries used: bindy.dll libximc.dll xiwrapper.dll. Located in the folder for the respective versions of Windows.

Launch Python 2 or Python 3:

```
python test_Python.py
```

In case of the 8SMC4-USB-Eth1 Ethernet adapter usage it is necessary to set correct IP address of the Ethernet adapter in test\_Python.py file before launch (see enum\_hints variable).

## 4.6 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 OS X: copy ximc/macosx/libximc.framework, ximc/macosx/wrappers/ximcm.h, ximc/ximc.h \* to the directory examples/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 compability 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/matlab. Then launch in MATLAB prompt:

```
testximc
```

In case of the 8SMC4-USB-Eth1 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).

## 4.7 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.

## 4.8 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.

## 4.9 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 "testcprofile" example directory.

## Chapter 5

# Working with custom units

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

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

### 5.1 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 custom units.

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

### 5.2 Alternative functions for working with custom units and data structures for them

Structures and functions for working with custom 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.

## 5.3 Coordinate correction table for more accurate positioning

Some functions for working with custom units support coordinate transformation using a correction table. To load a table from a file, the [load\\_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 6

# Data Structure Documentation

### 6.1 accessories\_settings\_t Struct Reference

Additional accessories information.

#### 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 (B).*
- 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.](#)*

#### 6.1.1 Detailed Description

Additional accessories information.

See Also

[set\\_accessories\\_settings](#)  
[get\\_accessories\\_settings](#)  
[get\\_accessories\\_settings](#), [set\\_accessories\\_settings](#)

## 6.1.2 Field Documentation

### 6.1.2.1 unsigned int LimitSwitchesSettings

[Temperature sensor settings flags.](#)

### 6.1.2.2 char MagneticBrakeInfo[25]

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

### 6.1.2.3 float MBRatedCurrent

Rated current for controlling the magnetic brake (A).

Data type: float.

### 6.1.2.4 float MBRatedVoltage

Rated voltage for controlling the magnetic brake (B).

Data type: float.

### 6.1.2.5 unsigned int MBSettings

[Magnetic brake settings flags.](#)

### 6.1.2.6 float MBTorque

Retention moment (mN m).

Data type: float.

### 6.1.2.7 char TemperatureSensorInfo[25]

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

### 6.1.2.8 float TSGrad

The temperature gradient (V/degrees Celsius).

Data type: float.

### 6.1.2.9 float TSMax

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

## 6.1.2.10 float TSMIn

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

## 6.1.2.11 unsigned int TSSettings

[Temperature sensor settings flags.](#)

## 6.2 analog\_data\_t Struct Reference

Analog data.

## 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](#)  
*"Voltage on the top of MOSFET full bridge" 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 [L5\\_ADC](#)  
*USB supply voltage after the current sense resistor, from ADC.*
- unsigned int [H5\\_ADC](#)  
*Power supply USB from ADC.*
- 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](#)  
*"Voltage on the top of MOSFET full bridge" 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 Celcius).*
- int [Joy](#)  
*Joystick, calibrated data.*
- int [Pot](#)  
*Analog input, calibrated data.*
- int [L5](#)  
*USB supply voltage after the current sense resistor (in tens of mV).*
- int [H5](#)  
*Power supply USB (in tens of mV).*
- unsigned int **deprecated**
- int [R](#)  
*Motor winding resistance in mOhms(is only used with stepper motor).*
- int [L](#)  
*Motor winding pseudo inductance in uHn(is only used with stepper motor).*

### 6.2.1 Detailed Description

Analog data.

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

See Also

[get\\_analog\\_data](#)  
[get\\_analog\\_data](#)

### 6.2.2 Field Documentation

#### 6.2.2.1 int A1Voltage

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

#### 6.2.2.2 unsigned int A1Voltage\_ADC

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

#### 6.2.2.3 int A2Voltage

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

6.2.2.4 unsigned int A2Voltage\_ADC

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

6.2.2.5 int ACurrent

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

6.2.2.6 unsigned int ACurrent\_ADC

"Winding A current" raw data from ADC.

6.2.2.7 int B1Voltage

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

6.2.2.8 unsigned int B1Voltage\_ADC

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

6.2.2.9 int B2Voltage

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

6.2.2.10 unsigned int B2Voltage\_ADC

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

6.2.2.11 int BCurrent

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

6.2.2.12 unsigned int BCurrent\_ADC

"Winding B current" raw data from ADC.

6.2.2.13 int FullCurrent

"Full current" calibrated data (in mA).

6.2.2.14 unsigned int FullCurrent\_ADC

"Full current" raw data from ADC.

6.2.2.15 int H5

Power supply USB (in tens of mV).

## 6.2.2.16 int Joy

Joystick, calibrated data.

Range: 0..10000

## 6.2.2.17 unsigned int Joy\_ADC

Joystick raw data from ADC.

## 6.2.2.18 int L

Motor winding pseudo inductance in uHn(is only used with stepper motor).

## 6.2.2.19 int L5

USB supply voltage after the current sense resistor (in tens of mV).

## 6.2.2.20 unsigned int L5\_ADC

USB supply voltage after the current sense resistor, from ADC.

## 6.2.2.21 int Pot

Analog input, calibrated data.

Range: 0..10000

## 6.2.2.22 int R

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

## 6.2.2.23 int SupVoltage

"Voltage on the top of MOSFET full bridge" calibrated data (in tens of mV).

## 6.2.2.24 unsigned int SupVoltage\_ADC

"Voltage on the top of MOSFET full bridge" raw data from ADC.

## 6.2.2.25 int Temp

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

## 6.2.2.26 unsigned int Temp\_ADC

Voltage from temperature sensor, raw data from ADC.

## 6.3 brake\_settings\_t Struct Reference

Brake settings.

### Data Fields

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

### 6.3.1 Detailed Description

Brake settings.

This structure contains parameters of brake control.

See Also

[set\\_brake\\_settings](#)  
[get\\_brake\\_settings](#)  
[get\\_brake\\_settings](#), [set\\_brake\\_settings](#)

### 6.3.2 Field Documentation

#### 6.3.2.1 unsigned int BrakeFlags

[Brake settings flags.](#)

#### 6.3.2.2 unsigned int t1

Time in ms between turn on motor power and turn off brake.

#### 6.3.2.3 unsigned int t2

Time in ms between turn off brake and moving readiness.

All moving commands will execute after this interval.

#### 6.3.2.4 unsigned int t3

Time in ms between motor stop and turn on brake.

#### 6.3.2.5 unsigned int t4

Time in ms between turn on brake and turn off motor power.

## 6.4 calibration\_settings\_t Struct Reference

Calibration settings.

### Data Fields

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

### 6.4.1 Detailed Description

Calibration settings.

This structure contains calibration settings.

See Also

[get\\_calibration\\_settings](#)  
[set\\_calibration\\_settings](#)  
[get\\_calibration\\_settings](#), [set\\_calibration\\_settings](#)

### 6.4.2 Field Documentation

#### 6.4.2.1 float CSS1.A

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

#### 6.4.2.2 float CSS1.B

Shift factor for the analogue measurements of the winding A current.

#### 6.4.2.3 float CSS2.A

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

#### 6.4.2.4 float CSS2.B

Shift factor for the analogue measurements of the winding B current.

## 6.4.2.5 float FullCurrent\_A

Scaling factor for the analogue measurements of the full current.

## 6.4.2.6 float FullCurrent\_B

Shift factor for the analogue measurements of the full current.

## 6.5 calibration\_t Struct Reference

Calibration companion structure.

### Data Fields

- double [A](#)  
*Multplier.*
- unsigned int [MicrostepMode](#)  
*Microstep mode.*

### 6.5.1 Detailed Description

Calibration companion structure.

## 6.6 chart\_data\_t Struct Reference

Additional device state.

### Data Fields

- int [WindingVoltageA](#)  
*In the case step motor, the voltage across the winding A (in tens of mV); in the case of a brushless, the voltage on the first coil, in the case of the only DC.*
- int [WindingVoltageB](#)  
*In the case step motor, the voltage across the winding B (in tens of mV); in case of a brushless, the voltage on the second winding, and in the case of DC is not used.*
- int [WindingVoltageC](#)  
*In the case of a brushless, the voltage on the third winding (in tens of mV), in the case step motor and DC is not used.*
- int [WindingCurrentA](#)  
*In the case step motor, the current in the coil A (in mA); brushless if the current in the first coil, and in the case of a single DC.*
- int [WindingCurrentB](#)  
*In the case step motor, the current in the coil B (in mA); brushless if the current in the second coil, and in the case of DC is not used.*
- int [WindingCurrentC](#)  
*In the case of a brushless, the current in the third winding (in mA), in the case step motor and DC is not used.*
- unsigned int [Pot](#)

- Analog input value in ten-thousandths.*
  - unsigned int [Joy](#)  
*The joystick position in the ten-thousandths.*
- int [DutyCycle](#)  
*Duty cycle of PWM.*

### 6.6.1 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](#)

### 6.6.2 Field Documentation

#### 6.6.2.1 int DutyCycle

Duty cycle of PWM.

#### 6.6.2.2 unsigned int Joy

The joystick position in the ten-thousandths.

Range: 0..10000

#### 6.6.2.3 unsigned int Pot

Analog input value in ten-thousandths.

Range: 0..10000

#### 6.6.2.4 int WindingCurrentA

In the case step motor, the current in the coil A (in mA); brushless if the current in the first coil, and in the case of a single DC.

#### 6.6.2.5 int WindingCurrentB

In the case step motor, the current in the coil B (in mA); brushless if the current in the second coil, and in the case of DC is not used.

#### 6.6.2.6 int WindingCurrentC

In the case of a brushless, the current in the third winding (in mA), in the case step motor and DC is not used.

## 6.6.2.7 int WindingVoltageA

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

## 6.6.2.8 int WindingVoltageB

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

## 6.6.2.9 int WindingVoltageC

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

## 6.7 control\_settings\_calb\_t Struct Reference

Control settings which use user units.

### Data Fields

- float [MaxSpeed](#) [10]  
*Array of speeds using with joystick and button control.*
- unsigned int [Timeout](#) [9]  
*timeout[i] is time in ms, after that max\_speed[i+1] is applying.*
- unsigned int [MaxClickTime](#)  
*Maximum click time (in ms).*
- unsigned int [Flags](#)  
*Control flags.*
- float [DeltaPosition](#)  
*Shift (delta) of position.*

### 6.7.1 Detailed Description

Control settings which use user units.

This structure contains control parameters. When choosing CTL\_MODE=1 switches motor control with the joystick. In this mode, the joystick to the maximum engine tends Move at MaxSpeed [i], where i=0 if the previous use This mode is not selected another i. Buttons switch the room rate i. When CTL\_MODE=2 is switched on motor control using the Left / right. When you click on the button motor starts to move in the appropriate direction at a speed MaxSpeed [0], at the end of time Timeout[i] motor move at a speed MaxSpeed [i+1]. at Transition from MaxSpeed [i] on MaxSpeed [i+1] to acceleration, as usual. The figure above shows the sensitivity of the joystick feature on its position.

See Also

[set\\_control\\_settings\\_calb](#)  
[get\\_control\\_settings\\_calb](#)  
[get\\_control\\_settings](#), [set\\_control\\_settings](#)

### 6.7.2 Field Documentation

#### 6.7.2.1 unsigned int Flags

Control flags.

#### 6.7.2.2 unsigned int MaxClickTime

Maximum click time (in ms).

Prior to the expiration of this time the first speed isn't enabled.

#### 6.7.2.3 float MaxSpeed[10]

Array of speeds using with joystick and button control.

#### 6.7.2.4 unsigned int Timeout[9]

timeout[i] is time in ms, after that max\_speed[i+1] is applying.

It is using with buttons control only.

## 6.8 control\_settings\_t Struct Reference

Control settings.

### Data Fields

- unsigned int [MaxSpeed](#) [10]  
*Array of speeds (full step) using with joystick and button control.*
- unsigned int [uMaxSpeed](#) [10]  
*Array of speeds (in microsteps) using with joystick and button control.*
- unsigned int [Timeout](#) [9]  
*timeout[i] is time in ms, after that max\_speed[i+1] is applying.*
- unsigned int [MaxClickTime](#)  
*Maximum click time (in ms).*
- unsigned int [Flags](#)  
*Control flags.*
- int [DeltaPosition](#)  
*Shift (delta) of position (full step)*
- int [uDeltaPosition](#)  
*Fractional part of the shift in micro steps.*

### 6.8.1 Detailed Description

Control settings.

This structure contains control parameters. When choosing CTL\_MODE=1 switches motor control with the joystick. In this mode, the joystick to the maximum engine tends Move at MaxSpeed [i], where i=0 if the previous use This mode is not selected another i. Buttons switch the room rate i. When CTL\_MODE=2

is switched on motor control using the Left / right. When you click on the button motor starts to move in the appropriate direction at a speed MaxSpeed [0], at the end of time Timeout[i] motor move at a speed MaxSpeed [i+1]. at Transition from MaxSpeed [i] on MaxSpeed [i+1] to acceleration, as usual. The figure above shows the sensitivity of the joystick feature on its position.

See Also

[set\\_control\\_settings](#)  
[get\\_control\\_settings](#)  
[get\\_control\\_settings](#), [set\\_control\\_settings](#)

## 6.8.2 Field Documentation

### 6.8.2.1 unsigned int Flags

[Control flags](#).

### 6.8.2.2 unsigned int MaxClickTime

Maximum click time (in ms).

Prior to the expiration of this time the first speed isn't enabled.

### 6.8.2.3 unsigned int MaxSpeed[10]

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

Range: 0..100000.

### 6.8.2.4 unsigned int Timeout[9]

timeout[i] is time in ms, after that max\_speed[i+1] is applying.

It is using with buttons control only.

### 6.8.2.5 int uDeltaPosition

Fractional part of the shift in micro steps.

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).

### 6.8.2.6 unsigned int uMaxSpeed[10]

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

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

## 6.9 controller\_name\_t Struct Reference

Controller user name and flags of setting.

## Data Fields

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

### 6.9.1 Detailed Description

Controller user name and flags of setting.

See Also

[get\\_controller\\_name](#), [set\\_controller\\_name](#)

### 6.9.2 Field Documentation

#### 6.9.2.1 char ControllerName[17]

User conroller name.

Can be set by user for his/her convinience. Max string length: 16 chars.

#### 6.9.2.2 unsigned int CtrlFlags

[Flags of internal controller settings.](#)

## 6.10 ctp\_settings\_t Struct Reference

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

## Data Fields

- unsigned int [CTPMinError](#)  
*Minimum contrast steps from step motor encoder position, wich set STATE\_CTP\_ERROR flag.*
- unsigned int [CTPFlags](#)  
*Position control flags.*

### 6.10.1 Detailed Description

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

When controlling the step motor with 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 (flag CTP\_ENABLED), the controller stores the current position in the footsteps of SM and the current position of the encoder. Further, at each step of the position encoder is converted into steps and if the difference is greater CTPMinError, a flag STATE\_CTP\_ERROR and set ALARM state. When controlling the step motor with speed sensor (CTP\_BASE 1), the position is controlled by him. The active edge of input clock controller stores the current value of steps. Further, at each turn checks how many steps shifted. When a mismatch CTPMinError a flag STATE\_CTP\_ERROR and set ALARM state.

See Also

[set\\_ctp\\_settings](#)  
[get\\_ctp\\_settings](#)  
[get\\_ctp\\_settings](#), [set\\_ctp\\_settings](#)

### 6.10.2 Field Documentation

#### 6.10.2.1 unsigned int CTPFlags

[Position control flags](#).

#### 6.10.2.2 unsigned int CTPMinError

Minimum contrast steps from step motor encoder position, wich set STATE\_CTP\_ERROR flag.

Measured in steps step motor.

## 6.11 debug\_read\_t Struct Reference

Debug data.

### Data Fields

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

### 6.11.1 Detailed Description

Debug data.

These data are used for device debugging by manufacturer only.

See Also

[get\\_debug\\_read](#)

### 6.11.2 Field Documentation

#### 6.11.2.1 `uint8_t` DebugData[128]

Arbitrary debug data.

## 6.12 debug\_write\_t Struct Reference

Debug data.

### Data Fields

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

### 6.12.1 Detailed Description

Debug data.

These data are used for device debugging by manufacturer only.

See Also

[set\\_debug\\_write](#)

### 6.12.2 Field Documentation

#### 6.12.2.1 uint8\_t DebugData[128]

Arbitrary debug data.

## 6.13 device\_information\_t Struct Reference

Read command controller information.

### 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](#)  
*Minor number of the hardware version.*
- unsigned int [Release](#)  
*Number of edits this release of hardware.*

### 6.13.1 Detailed Description

Read command controller information.

The controller responds to this command in any state. Manufacturer field for all XI\*\* devices should contain the string "XIMC" (validation is performed on it) The remaining fields contain information about the device.

See Also

[get\\_device\\_information](#)

[get\\_device\\_information\\_impl](#)

### 6.13.2 Field Documentation

#### 6.13.2.1 unsigned int Major

The major number of the hardware version.

## 6.13.2.2 unsigned int Minor

Minor number of the hardware version.

## 6.13.2.3 unsigned int Release

Number of edits this release of hardware.

## 6.14 device\_network\_information\_t Struct Reference

Device network information structure.

### 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)*

### 6.14.1 Detailed Description

Device network information structure.

## 6.15 edges\_settings\_calb\_t Struct Reference

Edges settings which use user units.

### 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.*

### 6.15.1 Detailed Description

Edges settings which use user units.

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

See Also

[set\\_edges\\_settings\\_calb](#)  
[get\\_edges\\_settings\\_calb](#)  
[get\\_edges\\_settings](#), [set\\_edges\\_settings](#)

### 6.15.2 Field Documentation

#### 6.15.2.1 unsigned int BorderFlags

[Border flags](#).

#### 6.15.2.2 unsigned int EnderFlags

[Limit switches flags](#).

#### 6.15.2.3 float LeftBorder

Left border position, used if BORDER\_IS\_ENCODER flag is set.

Corrected by the table.

#### 6.15.2.4 float RightBorder

Right border position, used if BORDER\_IS\_ENCODER flag is set.

Corrected by the table.

## 6.16 edges\_settings\_t Struct Reference

Edges settings.

### 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.*

### 6.16.1 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 lead to device malfunction, can lead to irreversible damage of board.

See Also

[set\\_edges\\_settings](#)  
[get\\_edges\\_settings](#)  
[get\\_edges\\_settings](#), [set\\_edges\\_settings](#)

### 6.16.2 Field Documentation

#### 6.16.2.1 unsigned int BorderFlags

[Border flags](#).

#### 6.16.2.2 unsigned int EnderFlags

[Limit switches flags](#).

#### 6.16.2.3 int LeftBorder

Left border position, used if BORDER\_IS\_ENCODER flag is set.

#### 6.16.2.4 int RightBorder

Right border position, used if BORDER\_IS\_ENCODER flag is set.

#### 6.16.2.5 int uLeftBorder

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

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

#### 6.16.2.6 int uRightBorder

Right border position in microsteps.

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

## 6.17 emf\_settings\_t Struct Reference

EMF settings.

## Data Fields

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

### 6.17.1 Detailed Description

EMF settings.

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

See Also

[set\\_emf\\_settings](#)  
[get\\_emf\\_settings](#)  
[get\\_emf\\_settings](#), [set\\_emf\\_settings](#)

### 6.17.2 Field Documentation

#### 6.17.2.1 unsigned int BackEMFFlags

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

#### 6.17.2.2 float Km

*Electromechanical ratio of the motor.*

#### 6.17.2.3 float L

*The inductance of the windings of the motor.*

#### 6.17.2.4 float R

*The resistance of the windings of the motor.*

## 6.18 encoder\_information\_t Struct Reference

Encoder information.

## Data Fields

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

## 6.18.1 Detailed Description

Encoder information.

See Also

[set\\_encoder\\_information](#)  
[get\\_encoder\\_information](#)  
[get\\_encoder\\_information](#), [set\\_encoder\\_information](#)

## 6.18.2 Field Documentation

## 6.18.2.1 char Manufacturer[17]

Manufacturer.

Max string length: 16 chars.

## 6.18.2.2 char PartNumber[25]

Series and PartNumber.

Max string length: 24 chars.

## 6.19 encoder\_settings\_t Struct Reference

Encoder settings.

## Data Fields

- float [MaxOperatingFrequency](#)  
*Max 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.*

### 6.19.1 Detailed Description

Encoder settings.

See Also

[set\\_encoder\\_settings](#)  
[get\\_encoder\\_settings](#)  
[get\\_encoder\\_settings](#), [set\\_encoder\\_settings](#)

### 6.19.2 Field Documentation

#### 6.19.2.1 unsigned int EncoderSettings

[Encoder settings flags](#).

#### 6.19.2.2 float MaxCurrentConsumption

Max current consumption (mA).

Data type: float.

#### 6.19.2.3 float MaxOperatingFrequency

Max operation frequency (kHz).

Data type: float.

#### 6.19.2.4 float SupplyVoltageMax

Maximum supply voltage (V).

Data type: float.

#### 6.19.2.5 float SupplyVoltageMin

Minimum supply voltage (V).

Data type: float.

## 6.20 engine\_advanced\_setup\_t Struct Reference

EAS settings.

### Data Fields

- unsigned int [stepcloseloop\\_Kw](#)  
*Mixing ratio of the actual and set speed, range [0, 100], default value 50.*
- unsigned int [stepcloseloop\\_Kp\\_low](#)  
*Position feedback in the low-speed zone, range [0, 65535], default value 1000.*
- unsigned int [stepcloseloop\\_Kp\\_high](#)  
*Position feedback in the high-speed zone, range [0, 65535], default value 33.*

### 6.20.1 Detailed Description

EAS settings.

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\\_advansed\\_setup](#)  
[get\\_engine\\_advansed\\_setup](#)  
[get\\_engine\\_advansed\\_setup](#), [set\\_engine\\_advansed\\_setup](#)

### 6.20.2 Field Documentation

#### 6.20.2.1 unsigned int stepcloseloop\_Kp\_high

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

#### 6.20.2.2 unsigned int stepcloseloop\_Kp\_low

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

#### 6.20.2.3 unsigned int stepcloseloop\_Kw

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

## 6.21 engine\_settings\_calb\_t Struct Reference

Movement limitations and settings, related to the motor, which use user units.

### 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).*

### 6.21.1 Detailed Description

Movement limitations and settings, related to the motor, which use user units.

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

See Also

[set\\_engine\\_settings\\_calb](#)  
[get\\_engine\\_settings\\_calb](#)  
[get\\_engine\\_settings](#), [set\\_engine\\_settings](#)

### 6.21.2 Field Documentation

#### 6.21.2.1 float Antiplay

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

Used if ENGINE\_ANTIPLAY flag is set.

#### 6.21.2.2 unsigned int EngineFlags

[Flags of engine settings.](#)

#### 6.21.2.3 unsigned int MicrostepMode

[Flags of microstep mode.](#)

#### 6.21.2.4 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

#### 6.21.2.5 float NomSpeed

Nominal speed.

Controller will keep motor speed below this value if ENGINE\_LIMIT\_RPM flag is set.

#### 6.21.2.6 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).

#### 6.21.2.7 unsigned int StepsPerRev

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

Range: 1..65535.

## 6.22 engine\_settings\_t Struct Reference

Movement limitations and settings, related to the motor.

### 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).*

### 6.22.1 Detailed Description

Movement limitations and settings, related to the motor.

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

See Also

[set\\_engine\\_settings](#)  
[get\\_engine\\_settings](#)  
[get\\_engine\\_settings](#), [set\\_engine\\_settings](#)

### 6.22.2 Field Documentation

#### 6.22.2.1 int Antiplay

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

Used if ENGINE\_ANTIPLAY flag is set.

#### 6.22.2.2 unsigned int EngineFlags

[Flags of engine settings.](#)

6.22.2.3 unsigned int MicrostepMode

[Flags of microstep mode.](#)

6.22.2.4 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

6.22.2.5 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.

6.22.2.6 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).

6.22.2.7 unsigned int StepsPerRev

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

Range: 1..65535.

6.22.2.8 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).

## 6.23 entype\_settings\_t Struct Reference

Engine type and driver type settings.

### Data Fields

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

### 6.23.1 Detailed Description

Engine type and driver type settings.

Parameters

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

See Also

[get\\_enttype\\_settings](#), [set\\_enttype\\_settings](#)

### 6.23.2 Field Documentation

#### 6.23.2.1 unsigned int `DriverType`

[Flags of driver type.](#)

#### 6.23.2.2 unsigned int `EngineType`

[Flags of engine type.](#)

## 6.24 `extended_settings_t` Struct Reference

EST settings.

Data Fields

- unsigned int **Param1**

### 6.24.1 Detailed Description

EST settings.

This structure EST.

See Also

[set\\_extended\\_settings](#)  
[get\\_extended\\_settings](#)  
[get\\_extended\\_settings](#), [set\\_extended\\_settings](#)

## 6.25 `extio_settings_t` Struct Reference

EXTIO settings.

## Data Fields

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

### 6.25.1 Detailed Description

EXTIO settings.

This structure contains all EXTIO settings. By default input event are signalled through rising front and output states are signalled by high logic state.

See Also

[get\\_extio\\_settings](#)  
[set\\_extio\\_settings](#)  
[get\\_extio\\_settings](#), [set\\_extio\\_settings](#)

### 6.25.2 Field Documentation

6.25.2.1 unsigned int EXTIOModeFlags

[External IO mode flags.](#)

6.25.2.2 unsigned int EXTIOSetupFlags

[External IO setup flags.](#)

## 6.26 feedback\_settings\_t Struct Reference

Feedback settings.

## 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.*

### 6.26.1 Detailed Description

Feedback settings.

This structure contains feedback settings.

See Also

[get\\_feedback\\_settings](#), [set\\_feedback\\_settings](#)

### 6.26.2 Field Documentation

#### 6.26.2.1 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.

#### 6.26.2.2 unsigned int FeedbackFlags

[Describes feedback flags.](#)

#### 6.26.2.3 unsigned int FeedbackType

[Feedback type.](#)

#### 6.26.2.4 unsigned int IPS

The 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.

## 6.27 gear\_information\_t Struct Reference

Gear information.

### Data Fields

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

### 6.27.1 Detailed Description

Gear information.

See Also

[set\\_gear\\_information](#)  
[get\\_gear\\_information](#)  
[get\\_gear\\_information](#), [set\\_gear\\_information](#)

### 6.27.2 Field Documentation

#### 6.27.2.1 char Manufacturer[17]

Manufacturer.

Max string length: 16 chars.

#### 6.27.2.2 char PartNumber[25]

Series and PartNumber.

Max string length: 24 chars.

## 6.28 gear\_settings\_t Struct Reference

Gear settings.

### Data Fields

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

### 6.28.1 Detailed Description

Gear settings.

See Also

[set\\_gear\\_settings](#)  
[get\\_gear\\_settings](#)  
[get\\_gear\\_settings](#), [set\\_gear\\_settings](#)

### 6.28.2 Field Documentation

#### 6.28.2.1 float Efficiency

Reduction gear efficiency (%).

Data type: float.

## 6.28.2.2 float InputInertia

Equivalent input gear inertia (g cm<sup>2</sup>).

Data type: float.

## 6.28.2.3 float MaxOutputBacklash

Output backlash of the reduction gear(degree).

Data type: float.

## 6.28.2.4 float RatedInputSpeed

Max speed on the input shaft (rpm).

Data type: float.

## 6.28.2.5 float RatedInputTorque

Max continuous torque (N m).

Data type: float.

## 6.28.2.6 float ReductionIn

Input reduction coefficient.

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

## 6.28.2.7 float ReductionOut

Output reduction coefficient.

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

## 6.29 get\_position\_calb\_t Struct Reference

Position information.

### Data Fields

- float [Position](#)  
*The position in the engine.*
- long\_t [EncPosition](#)  
*Encoder position.*

### 6.29.1 Detailed Description

Position information.

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

See Also

[get\\_position](#)

### 6.29.2 Field Documentation

#### 6.29.2.1 `long_t` `EncPosition`

Encoder position.

#### 6.29.2.2 `float` `Position`

The position in the engine.

Corrected by the table.

## 6.30 `get_position_t` Struct Reference

Position information.

### 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.*

### 6.30.1 Detailed Description

Position information.

Useful structure that contains position value in steps and micro for stepper motor and encoder steps of all engines.

See Also

[get\\_position](#)

### 6.30.2 Field Documentation

#### 6.30.2.1 `long_t` `EncPosition`

Encoder position.

#### 6.30.2.2 `int` `uPosition`

Microstep position is only used with stepper motors.

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

## 6.31 globally\_unique\_identifer\_t Struct Reference

Globally unique identifier.

### 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.*

### 6.31.1 Detailed Description

Globally unique identifier.

See Also

[get\\_globally\\_unique\\_identifer](#)

### 6.31.2 Field Documentation

#### 6.31.2.1 unsigned int UniqueID0

Unique ID 0.

#### 6.31.2.2 unsigned int UniqueID1

Unique ID 1.

#### 6.31.2.3 unsigned int UniqueID2

Unique ID 2.

#### 6.31.2.4 unsigned int UniqueID3

Unique ID 3.

## 6.32 hallsensor\_information\_t Struct Reference

Hall sensor information.

## Data Fields

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

## 6.32.1 Detailed Description

Hall sensor information.

See Also

[set\\_hallsensor\\_information](#)  
[get\\_hallsensor\\_information](#)  
[get\\_hallsensor\\_information](#), [set\\_hallsensor\\_information](#)

## 6.32.2 Field Documentation

## 6.32.2.1 char Manufacturer[17]

Manufacturer.

Max string length: 16 chars.

## 6.32.2.2 char PartNumber[25]

Series and PartNumber.

Max string length: 24 chars.

## 6.33 hallsensor\_settings\_t Struct Reference

Hall sensor settings.

## Data Fields

- float [MaxOperatingFrequency](#)  
*Max 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.*

### 6.33.1 Detailed Description

Hall sensor settings.

See Also

[set\\_hallsensor\\_settings](#)  
[get\\_hallsensor\\_settings](#)  
[get\\_hallsensor\\_settings](#), [set\\_hallsensor\\_settings](#)

### 6.33.2 Field Documentation

#### 6.33.2.1 float MaxCurrentConsumption

Max current consumption (mA).

Data type: float.

#### 6.33.2.2 float MaxOperatingFrequency

Max operation frequency (kHz).

Data type: float.

#### 6.33.2.3 float SupplyVoltageMax

Maximum supply voltage (V).

Data type: float.

#### 6.33.2.4 float SupplyVoltageMin

Minimum supply voltage (V).

Data type: float.

## 6.34 `home_settings_calb_t` Struct Reference

Position calibration settings which use user units.

### 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.*

### 6.34.1 Detailed Description

Position calibration settings which use user units.

This structure contains settings used in position calibrating. It specify behaviour of calibrating position.

See Also

[get\\_home\\_settings\\_calb](#)  
[set\\_home\\_settings\\_calb](#)  
[command\\_home](#)  
[get\\_home\\_settings](#), [set\\_home\\_settings](#)

### 6.34.2 Field Documentation

#### 6.34.2.1 float FastHome

Speed used for first motion.

#### 6.34.2.2 float HomeDelta

Distance from break point.

#### 6.34.2.3 unsigned int HomeFlags

[Home settings flags](#).

#### 6.34.2.4 float SlowHome

Speed used for second motion.

## 6.35 home\_settings\_t Struct Reference

Position calibration settings.

### Data Fields

- unsigned int [FastHome](#)  
*Speed used for first motion (full steps).*
- unsigned int [uFastHome](#)  
*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](#)  
*Part of the delta distance, microsteps.*
- unsigned int [HomeFlags](#)  
*Home settings flags.*

### 6.35.1 Detailed Description

Position calibration settings.

This structure contains settings used in position calibrating. It specify behaviour of calibrating position.

See Also

[get\\_home\\_settings](#)  
[set\\_home\\_settings](#)  
[command\\_home](#)  
[get\\_home\\_settings](#), [set\\_home\\_settings](#)

### 6.35.2 Field Documentation

#### 6.35.2.1 unsigned int FastHome

Speed used for first motion (full steps).

Range: 0..100000.

#### 6.35.2.2 int HomeDelta

Distance from break point (full steps).

#### 6.35.2.3 unsigned int HomeFlags

[Home settings flags](#).

#### 6.35.2.4 unsigned int SlowHome

Speed used for second motion (full steps).

Range: 0..100000.

#### 6.35.2.5 unsigned int uFastHome

Part of the speed for first motion, microsteps.

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

#### 6.35.2.6 int uHomeDelta

Part of the delta distance, microsteps.

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

#### 6.35.2.7 unsigned int uSlowHome

Part of the speed for second motion, microsteps.

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

## 6.36 `init_random_t` Struct Reference

Random key.

### Data Fields

- `uint8_t key` [16]  
*Random key.*

### 6.36.1 Detailed Description

Random key.

Structure that contains random key used in encryption of WKEY and SSER command contents.

See Also

[get\\_init\\_random](#)

### 6.36.2 Field Documentation

#### 6.36.2.1 `uint8_t key`[16]

Random key.

## 6.37 `joystick_settings_t` Struct Reference

Joystick settings.

### Data Fields

- unsigned int `JoyLowEnd`  
*Joystick lower end position.*
- unsigned int `JoyCenter`  
*Joystick center position.*
- unsigned int `JoyHighEnd`  
*Joystick higher end position.*
- unsigned int `ExpFactor`  
*Exponential nonlinearity factor.*
- unsigned int `DeadZone`  
*Joystick dead zone.*
- unsigned int `JoyFlags`  
*Joystick flags.*

### 6.37.1 Detailed Description

Joystick settings.

This structure contains joystick parameters. If joystick position is outside `DeadZone` limits from the central position a movement with speed, defined by the joystick `DeadZone` edge to 100% deviation, begins. Joystick

positions inside DeadZone limits correspond to zero speed (soft stop of motion) and positions beyond Low and High limits correspond MaxSpeed [i] or -MaxSpeed [i] (see command SCTL), where i = 0 by default and can be changed with left/right buttons (see command SCTL). If next speed in list is zero (both integer and microstep parts), the button press is ignored. First speed in list shouldn't be zero. The relationship between the deviation and the rate is exponential, allowing no switching speed combine high mobility and accuracy.

See Also

[set\\_joystick\\_settings](#)  
[get\\_joystick\\_settings](#)  
[get\\_joystick\\_settings](#), [set\\_joystick\\_settings](#)

## 6.37.2 Field Documentation

### 6.37.2.1 unsigned int DeadZone

Joystick dead zone.

### 6.37.2.2 unsigned int ExpFactor

Exponential nonlinearity factor.

### 6.37.2.3 unsigned int JoyCenter

Joystick center position.

Range: 0..10000.

### 6.37.2.4 unsigned int JoyFlags

[Joystick flags](#).

### 6.37.2.5 unsigned int JoyHighEnd

Joystick higher end position.

Range: 0..10000.

### 6.37.2.6 unsigned int JoyLowEnd

Joystick lower end position.

Range: 0..10000.

## 6.38 measurements\_t Struct Reference

The buffer holds no more than 25 points.

### Data Fields

- int [Speed](#) [25]

*Current speed in microsteps per second (whole steps are recalculated taking into account the current step division mode) or encoder counts per second.*

- int [Error](#) [25]

*Current error in microsteps per second (whole steps are recalculated taking into account the current step division mode) or encoder counts per second.*

- unsigned int [Length](#)

*Length of actual data in buffer.*

### 6.38.1 Detailed Description

The buffer holds no more than 25 points.

The exact length of the received buffer is reflected in the Length field.

See Also

measurements  
[get\\_measurements](#)

### 6.38.2 Field Documentation

#### 6.38.2.1 int Error[25]

Current error in microsteps per second (whole steps are recalculated taking into account the current step division mode) or encoder counts per second.

#### 6.38.2.2 unsigned int Length

Length of actual data in buffer.

#### 6.38.2.3 int Speed[25]

Current speed in microsteps per second (whole steps are recalculated taking into account the current step division mode) or encoder counts per second.

## 6.39 motor\_information\_t Struct Reference

motor information.

### Data Fields

- char [Manufacturer](#) [17]

*Manufacturer.*

- char [PartNumber](#) [25]

*Series and PartNumber.*

### 6.39.1 Detailed Description

motor information.

See Also

[set\\_motor\\_information](#)  
[get\\_motor\\_information](#)  
[get\\_motor\\_information](#), [set\\_motor\\_information](#)

## 6.39.2 Field Documentation

### 6.39.2.1 char Manufacturer[17]

Manufacturer.

Max string length: 16 chars.

### 6.39.2.2 char PartNumber[25]

Series and PartNumber.

Max string length: 24 chars.

## 6.40 motor\_settings\_t Struct Reference

Physical characteristics and limitations of the motor.

### 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 motor.*
- 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 motor (A).*
- float [NominalSpeed](#)  
*Not used.*
- float [NominalTorque](#)  
*Nominal torque(mN m).*
- float [NominalPower](#)  
*Nominal power(W).*
- float [WindingResistance](#)  
*Resistance of windings for DC engine, each of two windings for stepper motor or each of there windings for BLDC engine(Ohm).*
- float [WindingInductance](#)  
*Inductance of windings for DC engine, each of two windings for stepper motor or each of there windings for BLDC engine(mH).*

- float [RotorInertia](#)  
*Rotor inertia(g cm<sup>2</sup>).*
- 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 un-powered coils (mN m).*
- float [TorqueConstant](#)  
*Torque constant, which determines the aspect ratio of maximum moment of force from the rotor current flowing in the coil (mN m / A).*
- float [SpeedConstant](#)  
*Velocity constant, which determines the value or amplitude of the induced voltage on the motion of DC or BLDC motor (rpm / V) or stepper motor (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).*

### 6.40.1 Detailed Description

Physical characteristics and limitations of the motor.

See Also

[set\\_motor\\_settings](#)  
[get\\_motor\\_settings](#)  
[get\\_motor\\_settings](#), [set\\_motor\\_settings](#)

### 6.40.2 Field Documentation

#### 6.40.2.1 float DetentTorque

Holding torque position with un-powered coils (mN m).

Data type: float.

#### 6.40.2.2 float MaxCurrent

The maximum current in the winding (A).

Data type: float.

## 6.40.2.3 float MaxCurrentTime

Safe duration of overcurrent in the winding (ms).

Data type: float.

## 6.40.2.4 float MaxSpeed

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

Data type: float.

## 6.40.2.5 float MechanicalTimeConstant

Mechanical time constant (ms).

Data type: float.

## 6.40.2.6 unsigned int MotorType

[Motor Type flags](#).

## 6.40.2.7 float NoLoadCurrent

The current consumption in idle mode (A).

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

## 6.40.2.8 float NoLoadSpeed

Idle speed (rpm).

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

## 6.40.2.9 float NominalCurrent

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

Data type: float.

## 6.40.2.10 float NominalPower

Nominal power(W).

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

## 6.40.2.11 float NominalSpeed

Not used.

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

## 6.40.2.12 float NominalTorque

Nominal torque(mN m).

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

## 6.40.2.13 float NominalVoltage

Nominal voltage on winding (B).

Data type: float

## 6.40.2.14 unsigned int Phases

Number of phases for BLDC motors.

## 6.40.2.15 unsigned int Poles

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

## 6.40.2.16 float RotorInertia

Rotor inertia(g cm<sup>2</sup>).

Data type: float.

## 6.40.2.17 float SpeedConstant

Velocity constant, which determines the value or amplitude of the induced voltage on the motion of DC or BLDC motor (rpm / V) or stepper motor (steps/s / V).

Data type: float.

## 6.40.2.18 float SpeedTorqueGradient

Speed torque gradient (rpm / mN m).

Data type: float.

## 6.40.2.19 float StallTorque

Torque hold position for a stepper motor or torque at a motionless rotor for other types of engines (mN m).

Data type: float.

## 6.40.2.20 float TorqueConstant

Torque constant, which determines the aspect ratio of maximum moment of force from the rotor current flowing in the coil (mN m / A).

Used mainly for DC motors. Data type: float.

## 6.40.2.21 float WindingInductance

Inductance of windings for DC engine, each of two windings for stepper motor or each of there windings for BLDC engine(mH).

Data type: float.

## 6.40.2.22 float WindingResistance

Resistance of windings for DC engine, each of two windings for stepper motor or each of there windings for BLDC engine(Ohm).

Data type: float.

## 6.41 move\_settings\_calb\_t Struct Reference

Move settings which use user units.

### Data Fields

- float [Speed](#)  
*Target speed.*
- float [Accel](#)  
*Motor shaft acceleration, steps/s<sup>2</sup>(stepper motor) or RPM/s(DC).*
- float [Decel](#)  
*Motor shaft deceleration, steps/s<sup>2</sup>(stepper motor) or RPM/s(DC).*
- float [AntiplaySpeed](#)  
*Speed in antiplay mode.*
- unsigned int [MoveFlags](#)  
*Flags of the motion parameters.*

### 6.41.1 Detailed Description

Move settings which use user units.

See Also

[set\\_move\\_settings\\_calb](#)  
[get\\_move\\_settings\\_calb](#)  
[get\\_move\\_settings](#), [set\\_move\\_settings](#)

### 6.41.2 Field Documentation

## 6.41.2.1 float Accel

Motor shaft acceleration, steps/s<sup>2</sup>(stepper motor) or RPM/s(DC).

## 6.41.2.2 float AntiplaySpeed

Speed in antiplay mode.

## 6.41.2.3 float Decel

Motor shaft deceleration,  $\text{steps/s}^2$ (stepper motor) or  $\text{RPM/s}$ (DC).

## 6.41.2.4 unsigned int MoveFlags

[Flags of the motion parameters.](#)

## 6.41.2.5 float Speed

Target speed.

## 6.42 move\_settings\_t Struct Reference

Move settings.

### 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,  $\text{steps/s}^2$ (stepper motor) or  $\text{RPM/s}$ (DC).*
- unsigned int [Decel](#)  
*Motor shaft deceleration,  $\text{steps/s}^2$ (stepper motor) or  $\text{RPM/s}$ (DC).*
- unsigned int [AntiplaySpeed](#)  
*Speed in antiplay mode, full steps/s(stepper motor) or  $\text{RPM}$ (DC).*
- unsigned int [uAntiplaySpeed](#)  
*Speed in antiplay mode, microsteps/s.*
- unsigned int [MoveFlags](#)  
*[Flags of the motion parameters.](#)*

### 6.42.1 Detailed Description

Move settings.

See Also

[set\\_move\\_settings](#)  
[get\\_move\\_settings](#)  
[get\\_move\\_settings](#), [set\\_move\\_settings](#)

### 6.42.2 Field Documentation

## 6.42.2.1 unsigned int Accel

Motor shaft acceleration,  $\text{steps/s}^2$ (stepper motor) or  $\text{RPM/s}$ (DC).

Range: 1..65535.

## 6.42.2.2 unsigned int AntiplaySpeed

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

Range: 0..100000.

## 6.42.2.3 unsigned int Decel

Motor shaft deceleration, steps/s<sup>2</sup>(stepper motor) or RPM/s(DC).

Range: 1..65535.

## 6.42.2.4 unsigned int MoveFlags

[Flags of the motion parameters.](#)

## 6.42.2.5 unsigned int Speed

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

Range: 0..100000.

## 6.42.2.6 unsigned int uAntiplaySpeed

Speed in antiplay mode, microsteps/s.

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

## 6.42.2.7 unsigned int uSpeed

Target speed in microstep fractions/s.

Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine\_settings). Using with stepper motor only.

## 6.43 nonvolatile\_memory\_t Struct Reference

Userdata for save into FRAM.

### Data Fields

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

### 6.43.1 Detailed Description

Userdata for save into FRAM.

See Also

[get\\_nonvolatile\\_memory](#), [set\\_nonvolatile\\_memory](#)

### 6.43.2 Field Documentation

#### 6.43.2.1 unsigned int UserData[7]

User data.

Can be set by user for his/her convenience. 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 that all amd64 systems.

## 6.44 pid\_settings\_t Struct Reference

PID settings.

### 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.*

### 6.44.1 Detailed Description

PID settings.

This structure contains factors for PID routine. It specifies behaviour of PID routine for voltage. These factors are slightly different for different positioners. All boards are supplied with standard set of PID settings on 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](#)

## 6.45 power\_settings\_t Struct Reference

Step motor power settings.

## Data Fields

- unsigned int [HoldCurrent](#)  
*Current in holding regime, percent of nominal.*
- unsigned int [CurrReductDelay](#)  
*Time in ms from going to STOP state to reducing current.*
- unsigned int [PowerOffDelay](#)  
*Time in s from going to STOP state to turning power off.*
- unsigned int [CurrentSetTime](#)  
*Time in ms to reach nominal current.*
- unsigned int [PowerFlags](#)  
*Flags of power settings of stepper motor.*

## 6.45.1 Detailed Description

Step motor power settings.

See Also

[set\\_move\\_settings](#)  
[get\\_move\\_settings](#)  
[get\\_power\\_settings](#), [set\\_power\\_settings](#)

## 6.45.2 Field Documentation

6.45.2.1 unsigned int `CurrentSetTime`

Time in ms to reach nominal current.

6.45.2.2 unsigned int `CurrReductDelay`

Time in ms from going to STOP state to reducing current.

6.45.2.3 unsigned int `HoldCurrent`

Current in holding regime, percent of nominal.

Range: 0..100.

6.45.2.4 unsigned int `PowerFlags`

[Flags of power settings of stepper motor.](#)

6.45.2.5 unsigned int `PowerOffDelay`

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

6.46 `secure_settings_t` Struct Reference

This structure contains raw analog data from ADC embedded on board.

## Data Fields

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

## 6.46.1 Detailed Description

This structure contains raw analog data from ADC embedded on board.

These data used for device testing and deep recalibraton by manufacturer only.

See Also

[get\\_secure\\_settings](#)  
[set\\_secure\\_settings](#)  
[get\\_secure\\_settings](#), [set\\_secure\\_settings](#)

## 6.46.2 Field Documentation

6.46.2.1 unsigned int `Criticalpwr`

Maximum motor current which triggers ALARM state, in mA.

6.46.2.2 unsigned int `Criticallusb`

Maximum USB current which triggers ALARM state, in mA.

6.46.2.3 unsigned int `CriticalT`

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

6.46.2.4 unsigned int `CriticalUpwr`

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

6.46.2.5 unsigned int CriticalUusb

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

6.46.2.6 unsigned int Flags

[Flags of secure settings.](#)

6.46.2.7 unsigned int LowUpwrOff

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

6.46.2.8 unsigned int MinimumUusb

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

## 6.47 serial\_number\_t Struct Reference

Serial number structure and hardware version.

### 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](#)  
*Minor number of the hardware version.*
- unsigned int [Release](#)  
*Number of edits this release of hardware.*

### 6.47.1 Detailed Description

Serial number structure and hardware version.

The structure keep new serial number, hardware version and valid key. The SN and hardware version are changed and saved when transmitted key matches stored key. Can be used by manufacturer only.

See Also

[set\\_serial\\_number](#)

### 6.47.2 Field Documentation

6.47.2.1 uint8\_t Key[32]

Protection key (256 bit).

6.47.2.2 unsigned int Major

The major number of the hardware version.

6.47.2.3 unsigned int Minor

Minor number of the hardware version.

6.47.2.4 unsigned int Release

Number of edits this release of hardware.

6.47.2.5 unsigned int SN

New board serial number.

## 6.48 `set_position_calb_t` Struct Reference

Position information which use user units.

### Data Fields

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

### 6.48.1 Detailed Description

Position information which use user units.

Useful structure that contains position value in steps and micro for stepper motor and encoder steps of all engines.

See Also

[set\\_position](#)

### 6.48.2 Field Documentation

6.48.2.1 long\_t `EncPosition`

Encoder position.

6.48.2.2 unsigned int `PosFlags`

[Position setting flags.](#)

6.48.2.3 `float` Position

The position in the engine.

## 6.49 `set_position_t` Struct Reference

Position information.

### 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.*

### 6.49.1 Detailed Description

Position information.

Useful structure that contains position value in steps and micro for stepper motor and encoder steps of all engines.

See Also

[set\\_position](#)

### 6.49.2 Field Documentation

#### 6.49.2.1 `long_t` EncPosition

Encoder position.

#### 6.49.2.2 `unsigned int` PosFlags

[Position setting flags.](#)

#### 6.49.2.3 `int` uPosition

Microstep position is only used with stepper motors.

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

## 6.50 `stage_information_t` Struct Reference

Stage information.

## Data Fields

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

### 6.50.1 Detailed Description

Stage information.

See Also

[set\\_stage\\_information](#)  
[get\\_stage\\_information](#)  
[get\\_stage\\_information](#), [set\\_stage\\_information](#)

### 6.50.2 Field Documentation

#### 6.50.2.1 char Manufacturer[17]

Manufacturer.

Max string length: 16 chars.

#### 6.50.2.2 char PartNumber[25]

Series and PartNumber.

Max string length: 24 chars.

## 6.51 stage\_name\_t Struct Reference

Stage user name.

## Data Fields

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

### 6.51.1 Detailed Description

Stage user name.

See Also

[get\\_stage\\_name](#), [set\\_stage\\_name](#)

### 6.51.2 Field Documentation

#### 6.51.2.1 char PositionerName[17]

User positioner name.

Can be set by user for his/her convinience. Max string length: 16 chars.

## 6.52 stage\_settings\_t Struct Reference

Stage settings.

### 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](#)  
*Max speed (Units/c).*
- float [TravelRange](#)  
*Travel range (Units).*
- float [SupplyVoltageMin](#)  
*Supply voltage minimum (V).*
- float [SupplyVoltageMax](#)  
*Supply voltage maximum (V).*
- float [MaxCurrentConsumption](#)  
*Max current consumption (A).*
- float [HorizontalLoadCapacity](#)  
*Horizontal load capacity (kg).*
- float [VerticalLoadCapacity](#)  
*Vertical load capacity (kg).*

### 6.52.1 Detailed Description

Stage settings.

See Also

[set\\_stage\\_settings](#)  
[get\\_stage\\_settings](#)  
[get\\_stage\\_settings](#), [set\\_stage\\_settings](#)

### 6.52.2 Field Documentation

#### 6.52.2.1 float HorizontalLoadCapacity

Horizontal load capacity (kg).

Data type: float.

## 6.52.2.2 float LeadScrewPitch

Lead screw pitch (mm).

Data type: float.

## 6.52.2.3 float MaxCurrentConsumption

Max current consumption (A).

Data type: float.

## 6.52.2.4 float MaxSpeed

Max speed (Units/c).

Data type: float.

## 6.52.2.5 float SupplyVoltageMax

Supply voltage maximum (V).

Data type: float.

## 6.52.2.6 float SupplyVoltageMin

Supply voltage minimum (V).

Data type: float.

## 6.52.2.7 float TravelRange

Travel range (Units).

Data type: float.

## 6.52.2.8 char Units[9]

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

Max string length: 8 chars.

## 6.52.2.9 float VerticalLoadCapacity

Vertical load capacity (kg).

Data type: float.

## 6.53 status\_calb\_t Struct Reference

Device state which use user units.

## 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 [Ipwr](#)  
*Engine current, mA.*
- int [Upwr](#)  
*Power supply voltage, tens of mV.*
- int [Iusb](#)  
*USB current, mA.*
- int [Uusb](#)  
*USB voltage, tens of mV.*
- int [CurT](#)  
*Temperature in tenths of degrees C.*
- unsigned int [Flags](#)  
*Status flags.*
- unsigned int [GPIOFlags](#)  
*Status flags of the GPIO outputs.*
- unsigned int [CmdBufFreeSpace](#)  
*This field is a service field.*

## 6.53.1 Detailed Description

Device state which use user units.

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

See Also

`get_status_impl`

## 6.53.2 Field Documentation

## 6.53.2.1 unsigned int CmdBufFreeSpace

This field is a service field.

It shows the amount of free cells buffer synchronization chain.

## 6.53.2.2 float CurPosition

Current position.

Corrected by the table.

## 6.53.2.3 float CurSpeed

Motor shaft speed.

## 6.53.2.4 int CurT

Temperature in tenths of degrees C.

## 6.53.2.5 long\_t EncPosition

Current encoder position.

## 6.53.2.6 unsigned int EncSts

[Encoder state.](#)

## 6.53.2.7 unsigned int Flags

[Status flags.](#)

## 6.53.2.8 unsigned int GPIOFlags

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

## 6.53.2.9 int Ipwr

Engine current, mA.

## 6.53.2.10 int Iusb

USB current, mA.

## 6.53.2.11 unsigned int MoveSts

[Flags of move state.](#)

## 6.53.2.12 unsigned int MvCmdSts

[Move command state.](#)

## 6.53.2.13 unsigned int PWRSts

[Flags of power state of stepper motor.](#)

6.53.2.14 int Upwr

Power supply voltage, tens of mV.

6.53.2.15 int Uusb

USB voltage, tens of mV.

6.53.2.16 unsigned int WindSts

[Winding state.](#)

## 6.54 status\_t Struct Reference

Device state.

### 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](#)  
*Part of motor shaft speed in microsteps.*
- int [Ipwr](#)  
*Engine current, mA.*
- int [Upwr](#)  
*Power supply voltage, tens of mV.*
- int [Iusb](#)  
*USB current, mA.*
- int [Uusb](#)  
*USB voltage, tens of mV.*
- int [CurT](#)  
*Temperature in tenths of degrees C.*

- unsigned int [Flags](#)  
*Status flags.*
- unsigned int [GPIOFlags](#)  
*Status flags of the GPIO outputs.*
- unsigned int [CmdBufFreeSpace](#)  
*This field is a service field.*

### 6.54.1 Detailed Description

Device state.

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

See Also

`get_status_impl`

### 6.54.2 Field Documentation

#### 6.54.2.1 unsigned int CmdBufFreeSpace

This field is a service field.

It shows the amount of free cells buffer synchronization chain.

#### 6.54.2.2 int CurPosition

Current position.

#### 6.54.2.3 int CurSpeed

Motor shaft speed in steps/s or rpm.

#### 6.54.2.4 int CurT

Temperature in tenths of degrees C.

#### 6.54.2.5 long\_t EncPosition

Current encoder position.

#### 6.54.2.6 unsigned int EncSts

[Encoder state.](#)

#### 6.54.2.7 unsigned int Flags

[Status flags.](#)

6.54.2.8 unsigned int GPIOFlags

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

6.54.2.9 int Ipwr

Engine current, mA.

6.54.2.10 int Iusb

USB current, mA.

6.54.2.11 unsigned int MoveSts

[Flags of move state.](#)

6.54.2.12 unsigned int MvCmdSts

[Move command state.](#)

6.54.2.13 unsigned int PWRSts

[Flags of power state of stepper motor.](#)

6.54.2.14 int uCurPosition

Step motor shaft position in microsteps.

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

6.54.2.15 int uCurSpeed

Part of motor shaft speed in microsteps.

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

6.54.2.16 int Upwr

Power supply voltage, tens of mV.

6.54.2.17 int Uusb

USB voltage, tens of mV.

6.54.2.18 unsigned int WindSts

[Winding state.](#)

## 6.55 sync\_in\_settings\_calb\_t Struct Reference

Synchronization settings which use user units.

### Data Fields

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

### 6.55.1 Detailed Description

Synchronization settings which use user units.

This structure contains all synchronization settings, modes, periods and flags. It specifies behaviour of input synchronization. All boards are supplied with 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](#)

### 6.55.2 Field Documentation

#### 6.55.2.1 unsigned int ClutterTime

Input synchronization pulse dead time (mks).

#### 6.55.2.2 float Position

Desired position or shift.

#### 6.55.2.3 float Speed

Target speed.

#### 6.55.2.4 unsigned int SyncInFlags

[Flags for synchronization input setup.](#)

## 6.56 sync\_in\_settings\_t Struct Reference

Synchronization settings.

## Data Fields

- unsigned int [SyncInFlags](#)  
*Flags for synchronization input setup.*
- unsigned int [ClutterTime](#)  
*Input synchronization pulse dead time (mks).*
- 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.*

### 6.56.1 Detailed Description

Synchronization settings.

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

See Also

[get\\_sync\\_in\\_settings](#)  
[set\\_sync\\_in\\_settings](#)  
[get\\_sync\\_in\\_settings](#), [set\\_sync\\_in\\_settings](#)

### 6.56.2 Field Documentation

#### 6.56.2.1 unsigned int ClutterTime

Input synchronization pulse dead time (mks).

#### 6.56.2.2 unsigned int Speed

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

Range: 0..100000.

#### 6.56.2.3 unsigned int SyncInFlags

[Flags for synchronization input setup.](#)

#### 6.56.2.4 int uPosition

The fractional part of a position or shift in microsteps.

Is 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).

6.56.2.5 unsigned int uSpeed

Target speed in microsteps/s.

Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine\_settings). Using with stepper motor only.

## 6.57 sync\_out\_settings\_calb\_t Struct Reference

Synchronization settings which use user units.

### Data Fields

- unsigned int [SyncOutFlags](#)  
*Flags of synchronization output.*
- unsigned int [SyncOutPulseSteps](#)  
*This value specifies duration of output pulse.*
- unsigned int [SyncOutPeriod](#)  
*This value specifies 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 (in encoder pulses or motor steps), which is getting hit in the target position and the momentum generated by the stop.*

### 6.57.1 Detailed Description

Synchronization settings which use user units.

This structure contains all synchronization settings, modes, periods and flags. It specifies behaviour of output synchronization. All boards are supplied with 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](#)

### 6.57.2 Field Documentation

6.57.2.1 float Accuracy

This is the neighborhood around the target coordinates (in encoder pulses or motor steps), which is getting hit in the target position and the momentum generated by the stop.

6.57.2.2 unsigned int SyncOutFlags

[Flags of synchronization output.](#)

6.57.2.3 unsigned int SyncOutPeriod

This value specifies number of encoder pulses or steps between two output synchronization pulses when SYNCOUT\_ONPERIOD is set.

6.57.2.4 unsigned int SyncOutPulseSteps

This value specifies 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.

## 6.58 sync\_out\_settings\_t Struct Reference

Synchronization settings.

### Data Fields

- unsigned int [SyncOutFlags](#)  
*Flags of synchronization output.*
- unsigned int [SyncOutPulseSteps](#)  
*This value specifies duration of output pulse.*
- unsigned int [SyncOutPeriod](#)  
*This value specifies 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, which is getting hit in the target position and the momentum generated by the stop.*
- unsigned int [uAccuracy](#)  
*This is the neighborhood around the target coordinates in microsteps (only used with stepper motor).*

### 6.58.1 Detailed Description

Synchronization settings.

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

See Also

[get\\_sync\\_out\\_settings](#)  
[set\\_sync\\_out\\_settings](#)  
[get\\_sync\\_out\\_settings](#), [set\\_sync\\_out\\_settings](#)

### 6.58.2 Field Documentation

6.58.2.1 unsigned int Accuracy

This is the neighborhood around the target coordinates, which is getting hit in the target position and the momentum generated by the stop.

6.58.2.2 unsigned int SyncOutFlags

[Flags of synchronization output.](#)

6.58.2.3 unsigned int `SyncOutPeriod`

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

6.58.2.4 unsigned int `SyncOutPulseSteps`

This value specifies 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.

6.58.2.5 unsigned int `uAccuracy`

This is the neighborhood around the target coordinates in microsteps (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`).

## 6.59 `uart_settings_t` Struct Reference

UART settings.

### Data Fields

- unsigned int [Speed](#)  
*UART speed (in bauds)*
- unsigned int [UARTSetupFlags](#)  
*UART parity flags.*

### 6.59.1 Detailed Description

UART settings.

This structure contains UART settings.

See Also

[get\\_uart\\_settings](#)  
[set\\_uart\\_settings](#)  
[get\\_uart\\_settings](#), [set\\_uart\\_settings](#)

### 6.59.2 Field Documentation

6.59.2.1 unsigned int `UARTSetupFlags`

[UART parity flags.](#)

# Chapter 7

## File Documentation

### 7.1 ximc.h File Reference

Header file for libximc library.

#### Data Structures

- struct [calibration\\_t](#)  
*Calibration companion 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](#)  
*Move settings which use user units.*
- 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, which use user units.*
- 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 raw analog data from ADC embedded on board.*
- struct [edges\\_settings\\_t](#)  
*Edges settings.*
- struct [edges\\_settings\\_calb\\_t](#)  
*Edges settings which use user units.*
- struct [pid\\_settings\\_t](#)

- PID settings.*
- struct [sync\\_in\\_settings\\_t](#)
  - Synchronization settings.*
- struct [sync\\_in\\_settings\\_calb\\_t](#)
  - Synchronization settings which use user units.*
- 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](#)
  - Control settings which use user units.*
- struct [joystick\\_settings\\_t](#)
  - Joystick settings.*
- struct [ctp\\_settings\\_t](#)
  - Control position settings(is only used with stepper motor).*
- struct [uart\\_settings\\_t](#)
  - UART settings.*
- struct [calibration\\_settings\\_t](#)
  - Calibration settings.*
- struct [controller\\_name\\_t](#)
  - Controller user name and flags of setting.*
- struct [nonvolatile\\_memory\\_t](#)
  - Userdata for save into FRAM.*
- struct [emf\\_settings\\_t](#)
  - EMF settings.*
- struct [engine\\_advansed\\_setup\\_t](#)
  - EAS settings.*
- struct [extended\\_settings\\_t](#)
  - EST settings.*
- struct [get\\_position\\_t](#)
  - Position information.*
- struct [get\\_position\\_calb\\_t](#)
  - Position information.*
- struct [set\\_position\\_t](#)
  - Position information.*
- struct [set\\_position\\_calb\\_t](#)
  - Position information which use user units.*
- struct [status\\_t](#)
  - Device state.*
- struct [status\\_calb\\_t](#)
  - Device state which use user units.*
- struct [measurements\\_t](#)
  - The buffer holds no more than 25 points.*

- struct [chart\\_data\\_t](#)  
*Additional device state.*
- struct [device\\_information\\_t](#)  
*Read command controller information.*
- struct [serial\\_number\\_t](#)  
*Serial number structure and hardware version.*
- struct [analog\\_data\\_t](#)  
*Analog data.*
- struct [debug\\_read\\_t](#)  
*Debug data.*
- struct [debug\\_write\\_t](#)  
*Debug data.*
- struct [stage\\_name\\_t](#)  
*Stage user name.*
- struct [stage\\_information\\_t](#)  
*Stage information.*
- struct [stage\\_settings\\_t](#)  
*Stage settings.*
- struct [motor\\_information\\_t](#)  
*motor information.*
- struct [motor\\_settings\\_t](#)  
*Physical characteristics and limitations of the motor.*
- struct [encoder\\_information\\_t](#)  
*Encoder information.*
- struct [encoder\\_settings\\_t](#)  
*Encoder settings.*
- struct [hallsensor\\_information\\_t](#)  
*Hall sensor information.*
- struct [hallsensor\\_settings\\_t](#)  
*Hall sensor settings.*
- struct [gear\\_information\\_t](#)  
*Gear information.*
- struct [gear\\_settings\\_t](#)  
*Gear settings.*
- struct [accessories\\_settings\\_t](#)  
*Additional accessories information.*
- struct [init\\_random\\_t](#)  
*Random key.*
- struct [globally\\_unique\\_identifier\\_t](#)  
*Globally unique identifier.*

## Macros

- #define [XIMC\\_API](#)  
*Library import macro.*
- #define [XIMC\\_CALLCONV](#)  
*Library calling convention macros.*
- #define [XIMC\\_RETTYPE](#) void\*  
*Thread return type.*

- #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\\_noddevice](#) -4  
*device is lost*

### 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.*

### Enumerate devices flags

*This is a bit mask for bitwise operations.*

- #define [ENUMERATE\\_PROBE](#) 0x01  
*Check if a device with OS name name is 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 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 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  
*Flags of security.*
- #define [STATE\\_ALARM](#) 0x0000040  
*Controller is in alarm state indicating that something dangerous had 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  
*USB voltage exceeds safe limit.*
- #define [STATE\\_LOW\\_USB\\_VOLTAGE](#) 0x0002000  
*USB voltage is insufficient for normal operation.*
- #define [STATE\\_OVERLOAD\\_USB\\_CURRENT](#) 0x0004000  
*USB current exceeds safe limit.*
- #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 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 flag set; otherwise 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.*

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 rigt.*
- #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 have finished with an error, 0 - move command have finished correctly).*
- #define [MVCMD\\_RUNNING](#) 0x80  
*Move command state (0 - move command have finished, 1 - move command is being executed).*

### Flags of the motion parameters

*This is a bit mask for bitwise operations. Specify motor shaft movement algorithm and list of limitations. Flags returned by 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 milli rpm.*

### Flags of engine settings

*This is a bit mask for bitwise operations. Specify 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 of microstep mode. Using 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\\_DISCRETE\\_FET](#) 0x01  
*Driver with discrete FET keys.*
- #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 enabled after CurrReductDelay, if this flag is set.*
- #define [POWER\\_OFF\\_ENABLED](#) 0x02  
*Power off enabled after PowerOffDelay, if this flag is set.*
- #define [POWER\\_SMOOTH\\_CURRENT](#) 0x04  
*Current ramp-up/down is 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 Alarm state on driver overheat signal.*
- #define [LOW\\_UPWR\\_PROTECTION](#) 0x02  
*If this flag is set turn off motor when 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 [USB\\_BREAK\\_RECONNECT](#) 0x20  
*If this flag is set USB brake reconnect module will be enable.*
- #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 Alarm state on response of the engine control action.*

### Position setting flags

*This is a bit mask for bitwise operations. Flags used in setting of 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\\_TYPE\\_BITS](#) 0xC0  
*Bits of the encoder type.*
- #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.*

### 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 go to 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  
*Synchronization out pin follows the synchronization logic, if set.*
- #define [SYNCOUT\\_STATE](#) 0x02  
*When output state is fixed by negative SYNCOUT\_ENABLED flag, the pin state is in accordance with this flag state.*
- #define [SYNCOUT\\_INVERT](#) 0x04  
*Low level is active, if set, and high level is active otherwise.*
- #define [SYNCOUT\\_IN\\_STEPS](#) 0x08  
*Use motor steps/encoder pulses instead of milliseconds for output pulse generation if the flag is set.*
- #define [SYNCOUT\\_ONSTART](#) 0x10  
*Generate synchronization pulse when movement starts.*
- #define [SYNCOUT\\_ONSTOP](#) 0x20  
*Generate synchronization pulse when movement stops.*
- #define [SYNCOUT\\_ONPERIOD](#) 0x40  
*Generate 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 flag is set, works as input otherwise.*
- #define [EXTIO\\_SETUP\\_INVERT](#) 0x02  
*Interpret EXTIO states and fronts inverted if 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 behaviour 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 behaviour 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 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 behaviour 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 position on limit switches, if not set.*
- #define [BORDER\\_STOP\\_LEFT](#) 0x02  
*Motor should stop on 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 behaviour 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 behaviour 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 off power of step motor, 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 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 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 settings of position control. 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  
*Position control is enabled, if flag set.*
- #define [CTP\\_BASE](#) 0x02  
*Position control is based on revolution sensor, if this flag is set; otherwise it is based on encoder.*
- #define [CTP\\_ALARM\\_ON\\_ERROR](#) 0x04  
*Set ALARM on mismatch, if flag set.*
- #define [REV\\_SENS\\_INV](#) 0x08  
*Sensor is active when it 0 and invert makes active level 1.*
- #define [CTP\\_ERROR\\_CORRECTION](#) 0x10  
*Correct errors which appear when slippage if the flag is set.*

### Home settings flags

*This is a bit mask for bitwise operations. Specify behaviour for home command. 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  
*Flag defines direction of 1st motion after execution of home command.*
- #define [HOME\\_DIR\\_SECOND](#) 0x002  
*Flag defines direction of 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 in start of second movement the first half-turn.*
- #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 flag is set the encoder has differential output, else single ended output.*
- #define [ENCSET\\_PUSHPULL\\_OUTPUT](#) 0x004  
*If flag is set the encoder has push-pull output, else open drain output.*
- #define [ENCSET\\_INDEXCHANNEL\\_PRESENT](#) 0x010  
*If flag is set the encoder has index channel, else encoder hasn't it.*
- #define [ENCSET\\_REVOLUTIONSENSOR\\_PRESENT](#) 0x040  
*If flag is set the encoder has revolution sensor, else encoder hasn't it.*
- #define [ENCSET\\_REVOLUTIONSENSOR\\_ACTIVE\\_HIGH](#) 0x100  
*If flag is set the revolution sensor active state is high logic state, else 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 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  
*Unknow type of sensor.*
- #define [TS\\_TYPE\\_THERMOCOUPLE](#) 0x01  
*Thermocouple.*
- #define [TS\\_TYPE\\_SEMICONDUCTOR](#) 0x02  
*The semiconductor temperature sensor.*
- #define [TS\\_AVAILABLE](#) 0x08  
*If flag is set the temperature sensor is available.*
- #define [LS\\_ON\\_SW1\\_AVAILABLE](#) 0x01  
*If flag is set the limit switch connected to pin SW1 is available.*
- #define [LS\\_ON\\_SW2\\_AVAILABLE](#) 0x02  
*If flag is set the limit switch connected to pin SW2 is available.*
- #define [LS\\_SW1\\_ACTIVE\\_LOW](#) 0x04  
*If flag is set the limit switch connected to pin SW1 is triggered by a low level on pin.*
- #define [LS\\_SW2\\_ACTIVE\\_LOW](#) 0x08  
*If flag is set the limit switch connected to pin SW2 is triggered by a low level on pin.*
- #define [LS\\_SHORTED](#) 0x10  
*If flag is set the Limit switches is 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.*
- typedef struct [calibration\\_t](#) [calibration\\_t](#)  
*Calibration companion structure.*
- typedef struct  
[device\\_network\\_information\\_t](#) [device\\_network\\_information\\_t](#)  
*Device network information structure.*

## Functions

### Controller settings setup

Functions for adjusting engine read/write 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 home settings which use user units.*
- [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 home settings which use user units.*
- [result\\_t](#) [XIMC\\_API set\\_move\\_settings](#) ([device\\_t](#) id, const [move\\_settings\\_t](#) \*move\_settings)  
*Set command setup movement (speed, acceleration, threshold and etc).*
- [result\\_t](#) [XIMC\\_API set\\_move\\_settings\\_calb](#) ([device\\_t](#) id, const [move\\_settings\\_calb\\_t](#) \*move\_settings\_calb, const [calibration\\_t](#) \*calibration)  
*Set command setup movement which use user units (speed, acceleration, threshold and etc).*
- [result\\_t](#) [XIMC\\_API get\\_move\\_settings](#) ([device\\_t](#) id, [move\\_settings\\_t](#) \*move\_settings)

- Read command setup movement (speed, acceleration, threshold and etc).*
- [result\\_t XIMC\\_API get\\_move\\_settings\\_calb](#) ([device\\_t](#) id, [move\\_settings\\_calb\\_t](#) \*move\_settings\_calb, const [calibration\\_t](#) \*calibration)
- Read command setup movement which use user units (speed, acceleration, threshold and 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 engine settings which use user units.*
- [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 engine settings which use user units.*
- [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 which use 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 which use 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 synchronization settings which use user units.*
- [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 synchronization settings which use user units.*
- [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 synchronization settings which use user units.*
- [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 synchronization settings which use user units.*
- [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 settings of brake control.*
- [result\\_t XIMC\\_API get\\_brake\\_settings](#) ([device\\_t](#) id, [brake\\_settings\\_t](#) \*brake\_settings)
- Read settings of brake control.*
- [result\\_t XIMC\\_API set\\_control\\_settings](#) ([device\\_t](#) id, const [control\\_settings\\_t](#) \*control\_settings)
- Set settings of motor control.*
- [result\\_t XIMC\\_API set\\_control\\_settings\\_calb](#) ([device\\_t](#) id, const [control\\_settings\\_calb\\_t](#) \*control\_settings\_calb, const [calibration\\_t](#) \*calibration)
- Set settings of motor control which use user units.*
- [result\\_t XIMC\\_API get\\_control\\_settings](#) ([device\\_t](#) id, [control\\_settings\\_t](#) \*control\_settings)
- Read settings of motor control.*
- [result\\_t XIMC\\_API get\\_control\\_settings\\_calb](#) ([device\\_t](#) id, [control\\_settings\\_calb\\_t](#) \*control\_settings\_calb, const [calibration\\_t](#) \*calibration)
- Read settings of motor control which use user units.*
- [result\\_t XIMC\\_API set\\_joystick\\_settings](#) ([device\\_t](#) id, const [joystick\\_settings\\_t](#) \*joystick\_settings)
- Set settings of joystick.*
- [result\\_t XIMC\\_API get\\_joystick\\_settings](#) ([device\\_t](#) id, [joystick\\_settings\\_t](#) \*joystick\_settings)
- Read settings of joystick.*
- [result\\_t XIMC\\_API set\\_ctp\\_settings](#) ([device\\_t](#) id, const [ctp\\_settings\\_t](#) \*ctp\_settings)
- Set settings of control position(is only used with stepper motor).*
- [result\\_t XIMC\\_API get\\_ctp\\_settings](#) ([device\\_t](#) id, [ctp\\_settings\\_t](#) \*ctp\_settings)
- Read settings of control position(is only used with stepper motor).*
- [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\\_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 controller name and flags of setting from FRAM.*
- [result\\_t XIMC\\_API get\\_controller\\_name](#) ([device\\_t](#) id, [controller\\_name\\_t](#) \*controller\_name)
- Read user controller name and flags of setting from FRAM.*
- [result\\_t XIMC\\_API set\\_nonvolatile\\_memory](#) ([device\\_t](#) id, const [nonvolatile\\_memory\\_t](#) \*nonvolatile\_memory)
- Write userdata into FRAM.*
- [result\\_t XIMC\\_API get\\_nonvolatile\\_memory](#) ([device\\_t](#) id, [nonvolatile\\_memory\\_t](#) \*nonvolatile\_memory)
- Read userdata 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 stop the engine, the transition to the STOP, mode key BREAK (winding short-circuited), the regime "retention" is deactivated for DC motors, keeping current in the windings for stepper motors (with Power management settings).*
- `result_t XIMC_API command_power_off (device_t id)`  
*Immediately power off motor regardless its state.*
- `result_t XIMC_API command_move (device_t id, int Position, int uPosition)`  
*Upon receiving the command "move" the engine starts to move with pre-set parameters (speed, acceleration, retention), to the point specified to the Position, uPosition.*
- `result_t XIMC_API command_move_calb (device_t id, float Position, const calibration_t *calibration)`  
*Move to position which use user units.*
- `result_t XIMC_API command_movr (device_t id, int DeltaPosition, int uDeltaPosition)`  
*Move to offset.*
- `result_t XIMC_API command_movr_calb (device_t id, float DeltaPosition, const calibration_t *calibration)`  
*Move to offset using user units.*
- `result_t XIMC_API command_home (device_t id)`  
*The positive direction is to the right.*
- `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 point to a distance GENG :: Antiplay, then move to the same point.*
- `result_t XIMC_API command_sstp (device_t id)`  
*Soft stop engine.*
- `result_t XIMC_API get_position (device_t id, get_position_t *the_get_position)`  
*Reads the value position in steps and micro for stepper motor and encoder steps 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 all engines.*
- `result_t XIMC_API set_position (device_t id, const set_position_t *the_set_position)`  
*Sets any position value in steps and micro for stepper motor and encoder steps of all engines.*
- `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 which use user units.*
- `result_t XIMC_API command_zero (device_t id)`  
*Sets the current position and the position in which the traffic moves by the move command and movr zero for all cases, except for movement to the target position.*

### Group of commands to save and load settings

- `result_t XIMC_API command_save_settings (device_t id)`  
*Save all settings from controller's RAM to controller's flash memory, replacing previous data in controller's flash memory.*

- [result\\_t XIMC\\_API command\\_read\\_settings \(device\\_t id\)](#)  
*Read all settings from controller's flash memory to controller's RAM, replacing previous data in controller's RAM.*
- [result\\_t XIMC\\_API command\\_save\\_robust\\_settings \(device\\_t id\)](#)  
*Save important settings (calibration coefficients and etc.) from controller's RAM to controller's flash memory, replacing previous data in controller's flash memory.*
- [result\\_t XIMC\\_API command\\_read\\_robust\\_settings \(device\\_t id\)](#)  
*Read important settings (calibration coefficients and etc.) from controller's flash memory to controller's RAM, replacing previous data in controller's RAM.*
- [result\\_t XIMC\\_API command\\_eesave\\_settings \(device\\_t id\)](#)  
*Save settings from controller's RAM to stage's EEPROM memory, which spontaneity connected to stage and it isn't change without it mechanical reconstruction.*
- [result\\_t XIMC\\_API command\\_eeread\\_settings \(device\\_t id\)](#)  
*Read settings from controller's RAM to stage's EEPROM memory, which spontaneity connected to stage and it isn't change without it mechanical reconstruction.*
- [result\\_t XIMC\\_API command\\_start\\_measurements \(device\\_t id\)](#)  
*Start measurements and buffering of speed, following error.*
- [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 sequence error.*
- [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 controller's firmware version.*
- [result\\_t XIMC\\_API service\\_command\\_updf \(device\\_t id\)](#)  
*Command puts the controller to update the firmware.*

### 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 controller's flash memory.*
- [result\\_t XIMC\\_API get\\_analog\\_data \(device\\_t id, analog\\_data\\_t \\*analog\\_data\)](#)  
*Read analog data structure that contains raw analog data from ADC embedded on board.*
- [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.*

### Group of commands to work with EEPROM

- [result\\_t XIMC\\_API set\\_stage\\_name \(device\\_t id, const stage\\_name\\_t \\*stage\\_name\)](#)  
*Write user stage name from EEPROM.*
- [result\\_t XIMC\\_API get\\_stage\\_name \(device\\_t id, stage\\_name\\_t \\*stage\\_name\)](#)  
*Read user stage name from EEPROM.*
- [result\\_t XIMC\\_API set\\_stage\\_information \(device\\_t id, const stage\\_information\\_t \\*stage\\_information\)](#)  
*Set stage information to EEPROM.*
- [result\\_t XIMC\\_API get\\_stage\\_information \(device\\_t id, stage\\_information\\_t \\*stage\\_information\)](#)  
*Read stage information from EEPROM.*
- [result\\_t XIMC\\_API set\\_stage\\_settings \(device\\_t id, const stage\\_settings\\_t \\*stage\\_settings\)](#)  
*Set stage settings to EEPROM.*
- [result\\_t XIMC\\_API get\\_stage\\_settings \(device\\_t id, stage\\_settings\\_t \\*stage\\_settings\)](#)  
*Read stage settings from EEPROM.*
- [result\\_t XIMC\\_API set\\_motor\\_information \(device\\_t id, const motor\\_information\\_t \\*motor\\_information\)](#)

- Set motor information to EEPROM.*

  - [result\\_t XIMC\\_API get\\_motor\\_information](#) ([device\\_t](#) id, [motor\\_information\\_t](#) \*motor\_information)

*Read motor information from EEPROM.*
- [result\\_t XIMC\\_API set\\_motor\\_settings](#) ([device\\_t](#) id, const [motor\\_settings\\_t](#) \*motor\_settings)

*Set motor settings to EEPROM.*
- [result\\_t XIMC\\_API get\\_motor\\_settings](#) ([device\\_t](#) id, [motor\\_settings\\_t](#) \*motor\_settings)

*Read motor settings from EEPROM.*
- [result\\_t XIMC\\_API set\\_encoder\\_information](#) ([device\\_t](#) id, const [encoder\\_information\\_t](#) \*encoder\_information)

*Set encoder information to EEPROM.*
- [result\\_t XIMC\\_API get\\_encoder\\_information](#) ([device\\_t](#) id, [encoder\\_information\\_t](#) \*encoder\_information)

*Read encoder information from EEPROM.*
- [result\\_t XIMC\\_API set\\_encoder\\_settings](#) ([device\\_t](#) id, const [encoder\\_settings\\_t](#) \*encoder\_settings)

*Set encoder settings to EEPROM.*
- [result\\_t XIMC\\_API get\\_encoder\\_settings](#) ([device\\_t](#) id, [encoder\\_settings\\_t](#) \*encoder\_settings)

*Read encoder settings from EEPROM.*
- [result\\_t XIMC\\_API set\\_hallsensor\\_information](#) ([device\\_t](#) id, const [hallsensor\\_information\\_t](#) \*hallsensor\_information)

*Set hall sensor information to EEPROM.*
- [result\\_t XIMC\\_API get\\_hallsensor\\_information](#) ([device\\_t](#) id, [hallsensor\\_information\\_t](#) \*hallsensor\_information)

*Read hall sensor information from EEPROM.*
- [result\\_t XIMC\\_API set\\_hallsensor\\_settings](#) ([device\\_t](#) id, const [hallsensor\\_settings\\_t](#) \*hallsensor\_settings)

*Set hall sensor settings to EEPROM.*
- [result\\_t XIMC\\_API get\\_hallsensor\\_settings](#) ([device\\_t](#) id, [hallsensor\\_settings\\_t](#) \*hallsensor\_settings)

*Read hall sensor settings from EEPROM.*
- [result\\_t XIMC\\_API set\\_gear\\_information](#) ([device\\_t](#) id, const [gear\\_information\\_t](#) \*gear\_information)

*Set gear information to EEPROM.*
- [result\\_t XIMC\\_API get\\_gear\\_information](#) ([device\\_t](#) id, [gear\\_information\\_t](#) \*gear\_information)

*Read gear information from EEPROM.*
- [result\\_t XIMC\\_API set\\_gear\\_settings](#) ([device\\_t](#) id, const [gear\\_settings\\_t](#) \*gear\_settings)

*Set gear settings to EEPROM.*
- [result\\_t XIMC\\_API get\\_gear\\_settings](#) ([device\\_t](#) id, [gear\\_settings\\_t](#) \*gear\_settings)

*Read gear settings from EEPROM.*
- [result\\_t XIMC\\_API set\\_accessories\\_settings](#) ([device\\_t](#) id, const [accessories\\_settings\\_t](#) \*accessories\_settings)

*Set additional accessories information to EEPROM.*
- [result\\_t XIMC\\_API get\\_accessories\\_settings](#) ([device\\_t](#) id, [accessories\\_settings\\_t](#) \*accessories\_settings)

*Read additional accessories information from EEPROM.*
- [result\\_t XIMC\\_API get\\_bootloader\\_version](#) ([device\\_t](#) id, unsigned int \*Major, unsigned int \*Minor, unsigned int \*Release)

*Read controller's firmware version.*
- [result\\_t XIMC\\_API get\\_init\\_random](#) ([device\\_t](#) id, [init\\_random\\_t](#) \*init\_random)

*Read random number from 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 die 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.*

## 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 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 load\\_correction\\_table](#) (device\_t \*id, const char \*namefile)  
*Command of loading a correction table from a text file (this function is deprecated).*
- [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.*
- [result\\_t XIMC\\_API set\\_bindy\\_key](#) (const char \*keyfilepath)  
*Set network encryption layer (bindy) key.*
- [device\\_enumeration\\_t XIMC\\_API enumerate\\_devices](#) (int enumerate\_flags, const char \*hints)  
*Enumerate all devices that looks like valid.*
- [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.*

- [result\\_t XIMC\\_API ximc\\_fix\\_usbser\\_sys](#) (const char \*device\_uri)

*Fixing a USB driver error in Windows.*

- 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.*

### 7.1.1 Detailed Description

Header file for libximc library.

### 7.1.2 Macro Definition Documentation

#### 7.1.2.1 `#define ALARM_ON_DRIVER_OVERHEATING 0x01`

If this flag is set enter Alarm state on driver overheat signal.

#### 7.1.2.2 `#define BACK_EMF_INDUCTANCE_AUTO 0x01`

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

#### 7.1.2.3 `#define BACK_EMF_KM_AUTO 0x04`

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

#### 7.1.2.4 `#define BACK_EMF_RESISTANCE_AUTO 0x02`

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

7.1.2.5 `#define BORDER_IS_ENCODER 0x01`

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

7.1.2.6 `#define BORDER_STOP_LEFT 0x02`

Motor should stop on left border.

7.1.2.7 `#define BORDER_STOP_RIGHT 0x04`

Motor should stop on right border.

7.1.2.8 `#define BORDERS_SWAP_MISSET_DETECTION 0x08`

Motor should stop on both borders.

Need to save motor then wrong border settings is set

7.1.2.9 `#define BRAKE_ENABLED 0x01`

Brake control is enabled, if this flag is set.

7.1.2.10 `#define BRAKE_ENG_PWROFF 0x02`

Brake turns off power of step motor, if this flag is set.

7.1.2.11 `#define CONTROL_BTN_LEFT_PUSHED_OPEN 0x04`

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

7.1.2.12 `#define CONTROL_BTN_RIGHT_PUSHED_OPEN 0x08`

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

7.1.2.13 `#define CONTROL_MODE_BITS 0x03`

Bits to control engine by joystick or buttons.

7.1.2.14 `#define CONTROL_MODE_JOY 0x01`

Control by joystick.

7.1.2.15 `#define CONTROL_MODE_LR 0x02`

Control by left/right buttons.

7.1.2.16 `#define CONTROL_MODE_OFF 0x00`

Control is disabled.

7.1.2.17 `#define CTP_ALARM_ON_ERROR 0x04`

Set ALARM on mismatch, if flag set.

7.1.2.18 `#define CTP_BASE 0x02`

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

7.1.2.19 `#define CTP_ENABLED 0x01`

Position control is enabled, if flag set.

7.1.2.20 `#define CTP_ERROR_CORRECTION 0x10`

Correct errors which appear when slippage if the flag is set.

It works only with the encoder. Incompatible with flag CTP\_ALARM\_ON\_ERROR.

7.1.2.21 `#define DRIVER_TYPE_DISCRETE_FET 0x01`

Driver with discrete FET keys.

Default option.

7.1.2.22 `#define DRIVER_TYPE_EXTERNAL 0x03`

External driver.

7.1.2.23 `#define DRIVER_TYPE_INTEGRATE 0x02`

Driver with integrated IC.

7.1.2.24 `#define EEPROM_PRECEDENCE 0x01`

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

7.1.2.25 `#define ENC_STATE_ABSENT 0x00`

Encoder is absent.

7.1.2.26 `#define ENC_STATE_MALFUNC 0x02`

Encoder is connected and malfunctioning.

7.1.2.27 `#define ENC_STATE_OK 0x04`

Encoder is connected and working properly.

7.1.2.28 `#define ENC_STATE_REVERS 0x03`

Encoder is connected and operational but counts in other direction.

7.1.2.29 `#define ENC_STATE_UNKNOWN 0x01`

Encoder state is unknown.

7.1.2.30 `#define ENDER_SW1_ACTIVE_LOW 0x02`

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

7.1.2.31 `#define ENDER_SW2_ACTIVE_LOW 0x04`

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

7.1.2.32 `#define ENDER_SWAP 0x01`

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

7.1.2.33 `#define ENGINE_ACCEL_ON 0x10`

Acceleration enable flag.

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

7.1.2.34 `#define ENGINE_ANTIPLAY 0x08`

Play compensation flag.

If it set, engine makes backlash (play) compensation procedure and reach the predetermined position accurately on low speed.

7.1.2.35 `#define ENGINE_CURRENT_AS_RMS 0x02`

Engine current meaning flag.

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

7.1.2.36 `#define ENGINE_LIMIT_CURR 0x40`

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

7.1.2.37 `#define ENGINE_LIMIT_RPM 0x80`

Maximum motor speed limit enable flag.

7.1.2.38 `#define ENGINE_LIMIT_VOLT 0x20`

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

7.1.2.39 `#define ENGINE_MAX_SPEED 0x04`

Max speed flag.

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

7.1.2.40 `#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.

7.1.2.41 `#define ENGINE_TYPE_2DC 0x02`

2 DC motors.

7.1.2.42 `#define ENGINE_TYPE_BRUSHLESS 0x05`

Brushless motor.

7.1.2.43 `#define ENGINE_TYPE_DC 0x01`

DC motor.

7.1.2.44 `#define ENGINE_TYPE_NONE 0x00`

A value that shouldn't be used.

7.1.2.45 `#define ENGINE_TYPE_STEP 0x03`

Step motor.

7.1.2.46 `#define ENGINE_TYPE_TEST 0x04`

Duty cycle are fixed.

Used only manufacturer.

7.1.2.47 `#define ENUMERATE_PROBE 0x01`

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

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

7.1.2.48 `#define EXTIO_SETUP_INVERT 0x02`

Interpret EXTIO states and fronts inverted if flag is set.  
Falling front as input event and low logic level as active state.

7.1.2.49 `#define EXTIO_SETUP_MODE_IN_ALARM 0x05`

Set Alarm when the signal goes to the active state.

7.1.2.50 `#define EXTIO_SETUP_MODE_IN_BITS 0x0F`

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

7.1.2.51 `#define EXTIO_SETUP_MODE_IN_HOME 0x04`

Issue HOME command.

7.1.2.52 `#define EXTIO_SETUP_MODE_IN_MOVR 0x03`

Issue MOVR command with last used settings.

7.1.2.53 `#define EXTIO_SETUP_MODE_IN_NOP 0x00`

Do nothing.

7.1.2.54 `#define EXTIO_SETUP_MODE_IN_PWOF 0x02`

Issue PWOF command, powering off all engine windings.

7.1.2.55 `#define EXTIO_SETUP_MODE_IN_STOP 0x01`

Issue STOP command, ceasing the engine movement.

7.1.2.56 `#define EXTIO_SETUP_MODE_OUT_ALARM 0x30`

EXTIO pin stays active during Alarm state.

7.1.2.57 `#define EXTIO_SETUP_MODE_OUT_BITS 0xF0`

Bits of the output behaviour selection.

7.1.2.58 `#define EXTIO_SETUP_MODE_OUT_MOTOR_ON 0x40`

EXTIO pin stays active when windings are powered.

7.1.2.59 `#define EXTIO_SETUP_MODE_OUT_MOVING 0x20`

EXTIO pin stays active during moving state.

7.1.2.60 `#define EXTIO_SETUP_MODE_OUT_OFF 0x00`

EXTIO pin always set in inactive state.

7.1.2.61 `#define EXTIO_SETUP_MODE_OUT_ON 0x10`

EXTIO pin always set in active state.

7.1.2.62 `#define EXTIO_SETUP_OUTPUT 0x01`

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

7.1.2.63 `#define FEEDBACK_EMF 0x04`

Feedback by EMF.

7.1.2.64 `#define FEEDBACK_ENC_REVERSE 0x01`

Reverse count of encoder.

7.1.2.65 `#define FEEDBACK_ENC_TYPE_AUTO 0x00`

Auto detect encoder type.

7.1.2.66 `#define FEEDBACK_ENC_TYPE_BITS 0xC0`

Bits of the encoder type.

7.1.2.67 `#define FEEDBACK_ENC_TYPE_DIFFERENTIAL 0x80`

Differential encoder.

7.1.2.68 `#define FEEDBACK_ENC_TYPE_SINGLE_ENDED 0x40`

Single ended encoder.

7.1.2.69 `#define FEEDBACK_ENCODER 0x01`

Feedback by encoder.

7.1.2.70 `#define FEEDBACK_ENCODER_MEDIATED 0x06`

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

7.1.2.71 `#define FEEDBACK_NONE 0x05`

Feedback is absent.

7.1.2.72 `#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.

7.1.2.73 `#define HOME_DIR_FIRST 0x001`

Flag defines direction of 1st motion after execution of home command.

Direction is right, if set; otherwise left.

7.1.2.74 `#define HOME_DIR_SECOND 0x002`

Flag defines direction of 2nd motion.

Direction is right, if set; otherwise left.

7.1.2.75 `#define HOME_HALF_MV 0x008`

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

7.1.2.76 `#define HOME_MV_SEC_EN 0x004`

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

7.1.2.77 `#define HOME_STOP_FIRST_BITS 0x030`

Bits of the first stop selector.

7.1.2.78 `#define HOME_STOP_FIRST_LIM 0x030`

First motion stops by limit switch.

7.1.2.79 `#define HOME_STOP_FIRST_REV 0x010`

First motion stops by revolution sensor.

7.1.2.80 `#define HOME_STOP_FIRST_SYN 0x020`

First motion stops by synchronization input.

7.1.2.81 `#define HOME_STOP_SECOND_BITS 0x0C0`

Bits of the second stop selector.

7.1.2.82 `#define HOME_STOP_SECOND_LIM 0x0C0`

Second motion stops by limit switch.

7.1.2.83 `#define HOME_STOP_SECOND_REV 0x040`

Second motion stops by revolution sensor.

7.1.2.84 `#define HOME_STOP_SECOND_SYN 0x080`

Second motion stops by synchronization input.

7.1.2.85 `#define HOME_USE_FAST 0x100`

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

7.1.2.86 `#define JOY_REVERSE 0x01`

Joystick action is reversed.

Joystick deviation to the upper values correspond to negative speeds and vice versa.

7.1.2.87 `#define LOW_UPWR_PROTECTION 0x02`

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

7.1.2.88 `#define MICROSTEP_MODE_FRAC_128 0x08`

1/128 step mode.

7.1.2.89 `#define MICROSTEP_MODE_FRAC_16 0x05`

1/16 step mode.

7.1.2.90 `#define MICROSTEP_MODE_FRAC_2 0x02`

1/2 step mode.

7.1.2.91 `#define MICROSTEP_MODE_FRAC_256 0x09`

1/256 step mode.

7.1.2.92 `#define MICROSTEP_MODE_FRAC_32 0x06`

1/32 step mode.

7.1.2.93 `#define MICROSTEP_MODE_FRAC_4 0x03`

1/4 step mode.

7.1.2.94 `#define MICROSTEP_MODE_FRAC_64 0x07`

1/64 step mode.

7.1.2.95 `#define MICROSTEP_MODE_FRAC_8 0x04`

1/8 step mode.

7.1.2.96 `#define MICROSTEP_MODE_FULL 0x01`

Full step mode.

7.1.2.97 `#define MOVE_STATE_ANTIPLAY 0x04`

Motor is playing compensation, if flag set.

7.1.2.98 `#define MOVE_STATE_MOVING 0x01`

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

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

7.1.2.99 `#define MOVE_STATE_TARGET_SPEED 0x02`

Target speed is reached, if flag set.

7.1.2.100 `#define MVCMD_ERROR 0x40`

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

This flags is actual when `MVCMD_RUNNING` signals movement finish.

7.1.2.101 `#define MVCMD_HOME 0x06`

Command home.

7.1.2.102 `#define MVCMD_LEFT 0x03`

Command left.

7.1.2.103 `#define MVCMD_LOFT 0x07`

Command loft.

7.1.2.104 `#define MVCMD_MOVE 0x01`

Command move.

7.1.2.105 `#define MVCMD_MOVR 0x02`

Command movr.

7.1.2.106 `#define MVCMD_NAME_BITS 0x3F`

Move command bit mask.

7.1.2.107 `#define MVCMD_RIGHT 0x04`

Command right.

7.1.2.108 `#define MVCMD_RUNNING 0x80`

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

7.1.2.109 `#define MVCMD_SSTP 0x08`

Command soft stop.

7.1.2.110 `#define MVCMD_STOP 0x05`

Command stop.

7.1.2.111 `#define MVCMD_UKNWN 0x00`

Unknown command.

7.1.2.112 `#define POWER_OFF_ENABLED 0x02`

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

7.1.2.113 `#define POWER_REDUCT_ENABLED 0x01`

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

7.1.2.114 `#define POWER_SMOOTH_CURRENT 0x04`

Current ramp-up/down is performed smoothly during current\_set\_time, if this flag is set.

7.1.2.115 `#define PWR_STATE_MAX 0x05`

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

7.1.2.116 `#define PWR_STATE_NORM 0x03`

Motor windings are powered by nominal current.

7.1.2.117 `#define PWR_STATE_OFF 0x01`

Motor windings are disconnected from the driver.

7.1.2.118 `#define PWR.STATE.REDUCT 0x04`

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

7.1.2.119 `#define PWR.STATE.UNKNOWN 0x00`

Unknown state, should never happen.

7.1.2.120 `#define REV_SENS_INV 0x08`

Sensor is active when it 0 and invert makes active level 1.

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

7.1.2.121 `#define RPM_DIV_1000 0x01`

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

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

7.1.2.122 `#define SETPOS_IGNORE_ENCODER 0x02`

Will not reload encoder state if this flag is set.

7.1.2.123 `#define SETPOS_IGNORE_POSITION 0x01`

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

7.1.2.124 `#define STATE_ALARM 0x0000040`

Controller is in alarm state indicating that something dangerous had happened.

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

7.1.2.125 `#define STATE_BORDERS_SWAP_MISSET 0x0008000`

Engine stuck at the wrong edge.

7.1.2.126 `#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).

7.1.2.127 `#define STATE_BUTTON_LEFT 0x0008`

Button "left" state (1 if pressed).

7.1.2.128 `#define STATE_BUTTON_RIGHT 0x0004`

Button "right" state (1 if pressed).

7.1.2.129 `#define STATE_CONTR 0x000003F`

Flags of controller states.

7.1.2.130 `#define STATE_CONTROLLER_OVERHEAT 0x0000200`

Controller overheat.

7.1.2.131 `#define STATE_CTP_ERROR 0x0000080`

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

7.1.2.132 `#define STATE_DIG_SIGNAL 0xFFFF`

Flags of digital signals.

7.1.2.133 `#define STATE_EEPROM_CONNECTED 0x0000010`

EEPROM with settings is connected.

7.1.2.134 `#define STATE_ENC_A 0x2000`

State of encoder A pin.

7.1.2.135 `#define STATE_ENC_B 0x4000`

State of encoder B pin.

7.1.2.136 `#define STATE_ENGINE_RESPONSE_ERROR 0x0800000`

Error response of the engine control action.

7.1.2.137 `#define STATE_ERRC 0x0000001`

Command error encountered.

7.1.2.138 `#define STATE_ERRD 0x0000002`

Data integrity error encountered.

7.1.2.139 `#define STATE_ERRV 0x0000004`

Value error encountered.

7.1.2.140 `#define STATE_EXTIO_ALARM 0x1000000`

The error is caused by the input signal.

7.1.2.141 `#define STATE_GPIO_LEVEL 0x0020`

State of external GPIO pin.

7.1.2.142 `#define STATE_GPIO_PINOUT 0x0010`

External GPIO works as Out, if flag set; otherwise works as In.

7.1.2.143 `#define STATE_LEFT_EDGE 0x0002`

Engine stuck at the left edge.

7.1.2.144 `#define STATE_LOW_USB_VOLTAGE 0x0002000`

USB voltage is insufficient for normal operation.

7.1.2.145 `#define STATE_OVERLOAD_POWER_CURRENT 0x0000800`

Power current exceeds safe limit.

7.1.2.146 `#define STATE_OVERLOAD_POWER_VOLTAGE 0x0000400`

Power voltage exceeds safe limit.

7.1.2.147 `#define STATE_OVERLOAD_USB_CURRENT 0x0004000`

USB current exceeds safe limit.

7.1.2.148 `#define STATE_OVERLOAD_USB_VOLTAGE 0x0001000`

USB voltage exceeds safe limit.

7.1.2.149 `#define STATE_POWER_OVERHEAT 0x0000100`

Power driver overheat.

7.1.2.150 `#define STATE_REV_SENSOR 0x0400`

State of Revolution sensor pin.

7.1.2.151 `#define STATE_RIGHT_EDGE 0x0001`

Engine stuck at the right edge.

7.1.2.152 `#define STATE_SECUR 0x1B3FFC0`

Flags of security.

7.1.2.153 `#define STATE_SYNC_INPUT 0x0800`

State of Sync input pin.

7.1.2.154 `#define STATE_SYNC_OUTPUT 0x1000`

State of Sync output pin.

7.1.2.155 `#define SYNCIN_ENABLED 0x01`

Synchronization in mode is enabled, if this flag is set.

7.1.2.156 `#define SYNCIN_GOTOPOSITION 0x04`

The engine is go to position specified in Position and uPosition, if this flag is set.

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

7.1.2.157 `#define SYNCIN_INVERT 0x02`

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

7.1.2.158 `#define SYNCOUT_ENABLED 0x01`

Synchronization out pin follows the synchronization logic, if set.

It governed by SYNCOUT\_STATE flag otherwise.

7.1.2.159 `#define SYNCOUT_IN_STEPS 0x08`

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

7.1.2.160 `#define SYNCOUT_INVERT 0x04`

Low level is active, if set, and high level is active otherwise.

7.1.2.161 `#define SYNCOUT_ONPERIOD 0x40`

Generate synchronization pulse every SyncOutPeriod encoder pulses.

7.1.2.162 `#define SYNCOUT_ONSTART 0x10`

Generate synchronization pulse when movement starts.

7.1.2.163 `#define SYNCOUT_ONSTOP 0x20`

Generate synchronization pulse when movement stops.

7.1.2.164 `#define SYNCOUT_STATE 0x02`

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

7.1.2.165 `#define UART_PARITY_BITS 0x03`

Bits of the parity.

7.1.2.166 `#define WIND_A_STATE_ABSENT 0x00`

Winding A is disconnected.

7.1.2.167 `#define WIND_A_STATE_MALFUNC 0x02`

Winding A is short-circuited.

7.1.2.168 `#define WIND_A_STATE_OK 0x03`

Winding A is connected and working properly.

7.1.2.169 `#define WIND_A_STATE_UNKNOWN 0x01`

Winding A state is unknown.

7.1.2.170 `#define WIND_B_STATE_ABSENT 0x00`

Winding B is disconnected.

7.1.2.171 `#define WIND_B_STATE_MALFUNC 0x20`

Winding B is short-circuited.

7.1.2.172 `#define WIND_B_STATE_OK 0x30`

Winding B is connected and working properly.

7.1.2.173 `#define WIND_B_STATE_UNKNOWN 0x10`

Winding B state is unknown.

7.1.2.174 `#define XIMC_API`

Library import macro.

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

### 7.1.3 Typedef Documentation

7.1.3.1 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

### 7.1.4 Function Documentation

7.1.4.1 **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

7.1.4.2 **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
-----------	-------------------------

7.1.4.3 **result\_t XIMC\_API** command\_eeread\_settings ( **device\_t** id )

Read settings from controller's RAM to stage's EEPROM memory, which spontaneity connected to stage and it isn't change without it mechanical reconstruction.

Parameters

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

7.1.4.4 **result\_t XIMC\_API** command\_eesave\_settings ( **device\_t** id )

Save settings from controller's RAM to stage's EEPROM memory, which spontaneity connected to stage and it isn't change without it mechanical reconstruction.

Can be used by manufacturer only.

## Parameters

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

7.1.4.5 **result\_t XIMC\_API** command\_home ( **device\_t** id )

The positive direction is to the right.

A value of zero reverses the direction of the direction of the flag, the set speed. Restriction imposed by the trailer, act the same, except that the limit switch contact does not stop. Limit the maximum speed, acceleration and deceleration function. 1) moves the motor according to the speed FastHome, uFastHome and flag HOME\_DIR\_FAST until limit switch, if the flag is set HOME\_STOP\_ENDS, until the signal from the input synchronization if the flag HOME\_STOP\_SYNC (as accurately as possible is important to catch the moment of operation limit switch) or until the signal is received from the speed sensor, if the flag HOME\_STOP\_REV\_SN 2) then moves according to the speed SlowHome, uSlowHome and flag HOME\_DIR\_SLOW until signal from the clock input, if the flag HOME\_MV\_SEC. If the flag HOME\_MV\_SEC reset skip this paragraph. 3) then move the motor according to the speed FastHome, uFastHome and flag HOME\_DIR\_SLOW a distance HomeDelta, uHomeDelta. description of flags and variable see in description for commands GHOM/SHOM

## Parameters

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

## See Also

[home\\_settings\\_t](#)  
[get\\_home\\_settings](#)  
[set\\_home\\_settings](#)

7.1.4.6 **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.

7.1.4.7 **result\_t XIMC\_API** command\_left ( **device\_t** id )

Start continous moving to the left.

## Parameters

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

7.1.4.8 **result\_t XIMC\_API** command\_loft ( **device\_t** id )

Upon receiving the command "loft" the engine is shifted from the current point to a distance GENG :: Antiplay, then move to the same point.

Parameters

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

7.1.4.9 **result\_t XIMC\_API** command\_move ( **device\_t** id, int Position, int uPosition )

Upon receiving the command "move" the engine starts to move with pre-set parameters (speed, acceleration, retention), to the point specified to the Position, uPosition.

For stepper motor uPosition sets the microstep, for DC motor this field is not used.

Parameters

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

7.1.4.10 **result\_t XIMC\_API** command\_move\_calb ( **device\_t** id, float Position, const **calibration\_t** \* calibration )

Move to position which use user units.

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

Parameters

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

Note

The parameter Position is adjusted by the correction table.

7.1.4.11 **result\_t XIMC\_API** command\_movr ( **device\_t** id, int DeltaPosition, int uDeltaPosition )

Move to offset.

Upon receiving the command "movr" engine starts to move with pre-set parameters (speed, acceleration, hold), left or right (depending on the sign of DeltaPosition) by the number of pulses specified in the fields DeltaPosition, uDeltaPosition. For stepper motor uDeltaPosition sets the microstep, for DC motor this field is not used.

## Parameters

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

7.1.4.12 **result\_t XIMC\_API** command\_movr\_calb ( **device\_t** id, float DeltaPosition, const **calibration\_t** \* calibration )

Move to offset using user units.

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

## Parameters

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

## Note

The end coordinate is calculated using DeltaPosition, is adjusted by the correction table. To calculate coordinates correctly, when using a correction table, you do not need to execute movr commands in batches.

7.1.4.13 **result\_t XIMC\_API** command\_power\_off ( **device\_t** id )

Immediately power off motor regardless its state.

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

## Parameters

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

## See Also

[get\\_power\\_settings](#)  
[set\\_power\\_settings](#)

7.1.4.14 **result\_t XIMC\_API** command\_read\_robust\_settings ( **device\_t** id )

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

## Parameters

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

#### 7.1.4.15 **result\_t XIMC\_API** command\_read\_settings ( **device\_t** id )

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

Parameters

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

#### 7.1.4.16 **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
-----------	-------------------------

#### 7.1.4.17 **result\_t XIMC\_API** command\_right ( **device\_t** id )

Start continuous moving to the right.

Parameters

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

#### 7.1.4.18 **result\_t XIMC\_API** command\_save\_robust\_settings ( **device\_t** id )

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

Parameters

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

#### 7.1.4.19 **result\_t XIMC\_API** command\_save\_settings ( **device\_t** id )

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

Parameters

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

#### 7.1.4.20 **result\_t XIMC\_API** command\_sstp ( **device\_t** id )

Soft stop engine.

The motor stops with deceleration speed.

## Parameters

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

7.1.4.21 **result\_t** **XIMC\_API** command\_start\_measurements ( **device\_t** id )

Start measurements and buffering of speed, following error.

## Parameters

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

7.1.4.22 **result\_t** **XIMC\_API** command\_stop ( **device\_t** id )

Immediately stop the engine, the transition to the STOP, mode key BREAK (winding short-circuited), the regime "retention" is deactivated for DC motors, keeping current in the windings for stepper motors (with Power management settings).

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

## Parameters

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

7.1.4.23 **result\_t** **XIMC\_API** command\_update\_firmware ( const char \* uri, const uint8\_t \* data, uint32\_t data\_size )

Update firmware.

Service command

## Parameters

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

7.1.4.24 **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.
<i>out</i>	<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.

7.1.4.25 **result\_t XIMC\_API** command\_zero ( **device\_t** id )

Sets the current position and the position in which the traffic moves by the move command and movr zero for all cases, except for movement to the target position.

In the latter case, set the zero current position and the target position counted so that the absolute position of the destination is the same. That is, if we were at 400 and moved to 500, then the command Zero makes the current position of 0, and the position of the destination - 100. Does not change the mode of movement that is 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 device
-----------	-------------------------

7.1.4.26 **device\_enumeration\_t XIMC\_API** enumerate\_devices ( int enumerate\_flags, const char \* hints )

Enumerate all devices that looks like valid.

Parameters

<b>in</b>	<i>enumerate_flags</i>	enumerate devices flags
<b>in</b>	<i>hints</i>	extended information hints is a string of form "key=value \n key2=value2". Unrecognized key-value pairs are ignored. Key list: addr - used together with ENUMERATE_NETWORK flag. Non-null value is a remote host name or a comma-separated list of host names which contain the devices to be found, absent value means broadcast discovery. adapter_-addr - used together with ENUMERATE_NETWORK flag. Non-null value is a IP address of network adapter. Remote ximc device must be on the same local network as the adapter. When using the adapter_addr key, you must install the addr key. Example: "addr= \n adapter_addr=192.-168.0.100". To enumerate network devices you must call <a href="#">set_bindy_key</a> first.

7.1.4.27 **result\_t XIMC\_API** free\_enumerate\_devices ( **device\_enumeration\_t** device\_enumeration )

Free memory returned by *enumerate\_devices*.

Parameters

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

7.1.4.28 **result\_t XIMC\_API** get\_accessories\_settings ( **device\_t** id, **accessories\_settings\_t** \* accessories\_settings )

Read additional accessories information from EEPROM.

Parameters

	<i>id</i>	an identifier of device
<b>out</b>	<i>accessories_-settings</i>	structure contains information about additional accessories

7.1.4.29 **result\_t XIMC\_API** get\_analog\_data ( **device\_t** id, **analog\_data\_t** \* analog\_data )

Read analog data structure that contains raw analog data from ADC embedded on board.

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

Parameters

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

7.1.4.30 **result\_t XIMC\_API** get\_bootloader\_version ( **device\_t** id, unsigned int \* Major, unsigned int \* Minor, unsigned int \* Release )

Read controller's firmware version.

Parameters

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

7.1.4.31 **result\_t XIMC\_API** get\_brake\_settings ( **device\_t** id, **brake\_settings\_t** \* brake\_settings )

Read settings of brake control.

Parameters

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

7.1.4.32 **result\_t XIMC\_API** get\_calibration\_settings ( **device\_t** id, **calibration\_settings\_t** \* calibration\_settings )

Read calibration settings.

This function fill structure with calibration settings.

See Also

[calibration\\_settings\\_t](#)

Parameters

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

7.1.4.33 **result\_t XIMC\_API** get\_chart\_data ( **device\_t** id, **chart\_data\_t** \* chart\_data )

Return device electrical parameters, useful for charts.

Useful function that fill structure with snapshot of controller voltages and currents.

See Also

[chart\\_data\\_t](#)

Parameters

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

7.1.4.34 **result\_t XIMC\_API** get\_control\_settings ( **device\_t** id, **control\_settings\_t** \* control\_settings )

Read settings of motor control.

When choosing CTL\_MODE = 1 switches motor control with the joystick. In this mode, the joystick to the maximum engine tends Move at MaxSpeed [i], where i = 0 if the previous use This mode is not selected another i. Buttons switch the room rate i. When CTL\_MODE = 2 is switched on motor control using the Left / right. When you click on the button motor starts to move in the appropriate direction at a speed MaxSpeed [0], at the end of time Timeout [i] motor move at a speed MaxSpeed [i+1]. at Transition from MaxSpeed [i] on MaxSpeed [i +1] to acceleration, as usual.

Parameters

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

7.1.4.35 **result\_t XIMC\_API** get\_control\_settings\_calb ( **device\_t** id, **control\_settings\_calb\_t** \* control\_settings\_calb, const **calibration\_t** \* calibration )

Read settings of motor control which use user units.

When choosing CTL\_MODE = 1 switches motor control with the joystick. In this mode, the joystick to the maximum engine tends Move at MaxSpeed [i], where i = 0 if the previous use This mode is not selected another i. Buttons switch the room rate i. When CTL\_MODE = 2 is switched on motor control using the Left / right. When you click on the button motor starts to move in the appropriate direction at a speed MaxSpeed [0], at the end of time Timeout [i] motor move at a speed MaxSpeed [i+1]. at Transition from MaxSpeed [i] on MaxSpeed [i +1] to acceleration, as usual.

Parameters

	<i>id</i>	an identifier of device
out	<i>control_settings_calb</i>	structure contains settings motor control by joystick or buttons left/right.
	<i>calibration</i>	user unit settings

7.1.4.36 **result\_t XIMC\_API** get\_controller\_name ( **device\_t** id, **controller\_name\_t** \* controller\_name )

Read user controller name and flags of setting from FRAM.

Parameters

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

7.1.4.37 **result\_t XIMC\_API** get\_ctp\_settings ( **device\_t** id, **ctp\_settings\_t** \* ctp\_settings )

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

When controlling the step motor with 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 (flag CTP\_ENABLED), the controller stores the current position in the footsteps of SM and the current position of the encoder. Further, at each step of the position encoder is converted into steps and if the difference is greater CTPMinError, a flag STATE\_CTP\_ERROR. When controlling the step motor with speed sensor (CTP\_BASE 1), the position is controlled by him. The active edge of input clock controller stores the current value of steps. Further, at each turn checks how many steps shifted. When a mismatch CTPMinError a flag STATE\_CTP\_ERROR.

Parameters

	<i>id</i>	an identifier of device
out	<i>ctp_settings</i>	structure contains settings of control position

7.1.4.38 **result\_t XIMC\_API** get\_debug\_read ( **device\_t** id, **debug\_read\_t** \* debug\_read )

Read data from firmware for debug purpose.

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

Parameters

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

7.1.4.39 **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
----	---------------------------	--

7.1.4.40 **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](#)

7.1.4.41 **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

7.1.4.42 **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 device
out	<i>edges_settings</i>	edges settings, specify types of borders, motor behaviour and electrical behaviour of limit switches

7.1.4.43 **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 which use user units.

See Also

[set\\_edges\\_settings\\_calb](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>edges_settings_calb</i>	edges settings, specify types of borders, motor behaviour and electrical behaviour 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.

7.1.4.44 **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 device
out	<i>emf_settings</i>	EMF settings

7.1.4.45 **result\_t XIMC\_API** get\_encoder\_information ( **device\_t** id, **encoder\_information\_t** \* encoder\_information )

Read encoder information from EEPROM.

Parameters

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

7.1.4.46 **result\_t XIMC\_API** get\_encoder\_settings ( **device\_t** id, **encoder\_settings\_t** \* encoder\_settings )

Read encoder settings from EEPROM.

Parameters

	<i>id</i>	an identifier of device
out	<i>encoder_settings</i>	structure contains encoder settings

7.1.4.47 **result\_t XIMC\_API** get\_engine\_advanced\_setup ( **device\_t** id, **engine\_advanced\_setup\_t** \* engine\_advanced\_setup )

Read engine advanced settings.

See Also

[set\\_engine\\_advanced\\_setup](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>engine_advanced_setup</i>	EAS settings

7.1.4.48 **result\_t XIMC\_API** get\_engine\_settings ( **device\_t** id, **engine\_settings\_t** \* engine\_settings )

Read engine settings.

This function fill structure with set of useful motor settings stored in 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 device
out	<i>engine_settings</i>	engine settings

7.1.4.49 **result\_t XIMC\_API** get\_engine\_settings\_calb ( **device\_t** id, **engine\_settings\_calb\_t** \* engine\_settings\_calb, const **calibration\_t** \* calibration )

Read engine settings which use user units.

This function fill structure with set of useful motor settings stored in 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 device
out	<i>engine_settings_calb</i>	engine settings
	<i>calibration</i>	user unit settings

7.1.4.50 **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 device
out	<i>entype_settings</i>	structure contains settings motor type and power driver type

7.1.4.51 **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

7.1.4.52 **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

7.1.4.53 **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

7.1.4.54 **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

7.1.4.55 **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

7.1.4.56 **result\_t XIMC\_API** get\_extended\_settings ( **device\_t** id, **extended\_settings\_t** \* extended\_settings )

Read extended settings.

See Also

[set\\_extended\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>extended_settings</i>	EST settings

7.1.4.57 **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 controller's memory.

See Also

[set\\_extio\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>extio_settings</i>	EXTIO settings

7.1.4.58 **result\_t XIMC\_API** get\_feedback\_settings ( **device\_t** id, **feedback\_settings\_t** \* feedback\_settings )

Feedback settings.

Parameters

	<i>id</i>	an identifier of 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.

7.1.4.59 **result\_t XIMC\_API** get\_firmware\_version ( **device\_t** id, unsigned int \* Major, unsigned int \* Minor, unsigned int \* Release )

Read controller's firmware version.

Parameters

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

7.1.4.60 **result\_t XIMC\_API** get\_gear\_information ( **device\_t** id, **gear\_information\_t** \* gear\_information )

Read gear information from EEPROM.

Parameters

	<i>id</i>	an identifier of device
out	<i>gear-information</i>	structure contains information about step gearhead

7.1.4.61 **result\_t XIMC\_API** get\_gear\_settings ( **device\_t** id, **gear\_settings\_t** \* gear\_settings )

Read gear settings from EEPROM.

Parameters

	<i>id</i>	an identifier of device
out	<i>gear_settings</i>	structure contains step gearhead settings

7.1.4.62 **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 die but is not a random value.

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 device
out	<i>globally_unique_identifier</i>	the result of fields 0-3 concatenated defines the unique 128-bit device identifier.

7.1.4.63 **result\_t XIMC\_API** get\_hallsensor\_information ( **device\_t** id, **hallsensor\_information\_t** \* hallsensor\_information )

Read hall sensor information from EEPROM.

## Parameters

	<i>id</i>	an identifier of device
out	<i>hallsensor_information</i>	structure contains information about hall sensor

7.1.4.64 **result\_t XIMC\_API** get\_hallsensor\_settings ( **device\_t** id, **hallsensor\_settings\_t** \* hallsensor\_settings )

Read hall sensor settings from EEPROM.

## Parameters

	<i>id</i>	an identifier of device
out	<i>hallsensor_settings</i>	structure contains hall sensor settings

7.1.4.65 **result\_t XIMC\_API** get\_home\_settings ( **device\_t** id, **home\_settings\_t** \* home\_settings )

Read home settings.

This function fill structure with settings of calibrating position.

See Also

[home\\_settings\\_t](#)

## Parameters

	<i>id</i>	an identifier of device
out	<i>home_settings</i>	calibrating position settings

7.1.4.66 **result\_t XIMC\_API** get\_home\_settings\_calb ( **device\_t** id, **home\_settings\_calb\_t** \* home\_settings\_calb, const **calibration\_t** \* calibration )

Read home settings which use user units.

This function fill structure with settings of calibrating position.

See Also

[home\\_settings\\_calb\\_t](#)

## Parameters

	<i>id</i>	an identifier of device
out	<i>home_settings_calb</i>	calibrating position settings
	<i>calibration</i>	user unit settings

7.1.4.67 **result\_t XIMC\_API** get\_init\_random ( **device\_t** id, **init\_random\_t** \* init\_random )


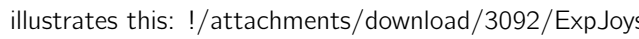
Read random number from controller.

## Parameters

	<i>id</i>	an identifier of device
out	<i>init_random</i>	random sequence generated by the controller

7.1.4.68 **result\_t XIMC\_API** get\_joystick\_settings ( **device\_t** id, **joystick\_settings\_t** \* joystick\_settings )

Read settings of joystick.

If joystick position is outside DeadZone limits from the central position a movement with speed, defined by the joystick DeadZone edge to 100% deviation, begins. Joystick positions inside DeadZone limits correspond to zero speed (soft stop of motion) and positions beyond Low and High limits correspond MaxSpeed [i] or -MaxSpeed [i] (see command SCTL), where  $i = 0$  by default and can be changed with left/right buttons (see command SCTL). If next speed in list is zero (both integer and microstep parts), the button press is ignored. First speed in list shouldn't be zero. The DeadZone ranges are illustrated on the following picture.  The relationship between the deviation and the rate is exponential, allowing no switching speed combine high mobility and accuracy. The following picture illustrates this:  The nonlinearity parameter is adjustable. Setting it to zero makes deviation/speed relation linear.

## Parameters

	<i>id</i>	an identifier of device
out	<i>joystick_settings</i>	structure contains joystick settings

7.1.4.69 **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 sequence error.

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 completely full, then it is recommended to repeat the readings every 5 ms until the buffer again becomes filled with 20 points.

See Also

[measurements\\_t](#)

## Parameters

	<i>id</i>	an identifier of device
out	<i>measurements</i>	structure with buffer and its length.

7.1.4.70 **result\_t XIMC\_API** get\_motor\_information ( **device\_t** id, **motor\_information\_t** \* motor\_information )

Read motor information from EEPROM.

Parameters

	<i>id</i>	an identifier of device
out	<i>motor_information</i>	structure contains motor information

7.1.4.71 **result\_t XIMC\_API** get\_motor\_settings ( **device\_t** id, **motor\_settings\_t** \* motor\_settings )

Read motor settings from EEPROM.

Parameters

	<i>id</i>	an identifier of device
out	<i>motor_settings</i>	structure contains motor settings

7.1.4.72 **result\_t XIMC\_API** get\_move\_settings ( **device\_t** id, **move\_settings\_t** \* move\_settings )

Read command setup movement (speed, acceleration, threshold and etc).

Parameters

	<i>id</i>	an identifier of device
out	<i>move_settings</i>	structure contains move settings: speed, acceleration, deceleration etc.

7.1.4.73 **result\_t XIMC\_API** get\_move\_settings\_calb ( **device\_t** id, **move\_settings\_calb\_t** \* move\_settings\_calb, const **calibration\_t** \* calibration )

Read command setup movement which use user units (speed, acceleration, threshold and etc).

Parameters

	<i>id</i>	an identifier of device
out	<i>move_settings_calb</i>	structure contains move settings: speed, acceleration, deceleration etc.
	<i>calibration</i>	user unit settings

7.1.4.74 **result\_t XIMC\_API** get\_nonvolatile\_memory ( **device\_t** id, **nonvolatile\_memory\_t** \* nonvolatile\_memory )

Read userdata from FRAM.

Parameters

	<i>id</i>	an identifier of device
out	<i>nonvolatile_memory</i>	structure contains previously set userdata

7.1.4.75 **result\_t XIMC\_API** get\_pid\_settings ( **device\_t** id, **pid\_settings\_t** \* pid\_settings )

Read PID settings.

This function fill structure with set of motor PID settings stored in controller's memory. These settings specify behaviour of PID routine for positioner. These factors are slightly different for different positioners. All boards are supplied with standard set of PID setting on controller's flash memory.

See Also

[set\\_pid\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>pid_settings</i>	pid settings

7.1.4.76 **result\_t XIMC\_API** get\_position ( **device\_t** id, **get\_position\_t** \* the\_get\_position )

Reads the value position in steps and micro for stepper motor and encoder steps all engines.

Parameters

	<i>id</i>	an identifier of device
out	<i>the_get_position</i>	structure contains move settings: speed, acceleration, deceleration etc.

7.1.4.77 **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 all engines.

Parameters

	<i>id</i>	an identifier of device
out	<i>the_get_position_calb</i>	structure contains move settings: speed, acceleration, deceleration etc.
	<i>calibration</i>	user unit settings

Note

Attention! Some parameters of the the\_get\_position\_calb structure are corrected by the coordinate correction table.

7.1.4.78 **result\_t XIMC\_API** get\_power\_settings ( **device\_t** id, **power\_settings\_t** \* power\_settings )

Read settings of step motor power control.

Used with stepper motor only.

Parameters

	<i>id</i>	an identifier of device
out	<i>power_settings</i>	structure contains settings of step motor power control

7.1.4.79 **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 device
out	<i>secure_settings</i>	critical parameter settings to protect the hardware

See Also

status\_t::flags

7.1.4.80 **result\_t XIMC\_API** get\_serial\_number ( **device\_t** id, unsigned int \* SerialNumber )

Read device serial number.

Parameters

	<i>id</i>	an identifier of device
out	<i>SerialNumber</i>	serial number

7.1.4.81 **result\_t XIMC\_API** get\_stage\_information ( **device\_t** id, **stage\_information\_t** \* stage\_information )

Read stage information from EEPROM.

Parameters

	<i>id</i>	an identifier of device
out	<i>stage-information</i>	structure contains stage information

7.1.4.82 **result\_t XIMC\_API** get\_stage\_name ( **device\_t** id, **stage\_name\_t** \* stage\_name )

Read user stage name from EEPROM.

Parameters

	<i>id</i>	an identifier of device
out	<i>stage_name</i>	structure contains previously set user stage name

7.1.4.83 **result\_t XIMC\_API** get\_stage\_settings ( **device\_t** id, **stage\_settings\_t** \* stage\_settings )

Read stage settings from EEPROM.

Parameters

	<i>id</i>	an identifier of device
out	<i>stage_settings</i>	structure contains stage settings

7.1.4.84 **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](#)

7.1.4.85 **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](#)

7.1.4.86 **result\_t XIMC\_API** get\_sync\_in\_settings ( **device\_t** id, **sync\_in\_settings\_t** \* sync\_in\_settings )

Read input synchronization settings.

This function fill structure with set of input synchronization settings, modes, periods and flags, that specify behaviour of input synchronization. All boards are supplied with standard set of these settings.

See Also

[set\\_sync\\_in\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>sync_in_settings</i>	synchronization settings

7.1.4.87 **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 synchronization settings which use user units.

This function fill structure with set of input synchronization settings, modes, periods and flags, that specify behaviour of input synchronization. All boards are supplied with standard set of these settings.

See Also

[set\\_sync\\_in\\_settings\\_calb](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>sync_in_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

7.1.4.88 **result\_t XIMC\_API** get\_sync\_out\_settings ( **device\_t** id, **sync\_out\_settings\_t** \* sync\_out\_settings )

Read output synchronization settings.

This function fill structure with set of output synchronization settings, modes, periods and flags, that specify behaviour of output synchronization. All boards are supplied with standard set of these settings.

See Also

[set\\_sync\\_out\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>sync_out_settings</i>	synchronization settings

7.1.4.89 **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 synchronization settings which use user units.

This function fill structure with set of output synchronization settings, modes, periods and flags, that specify behaviour of output synchronization. All boards are supplied with standard set of these settings.

See Also

[set\\_sync\\_in\\_settings\\_calb](#)

Parameters

	<i>id</i>	an identifier of device
out	<i>sync_out_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

7.1.4.90 **result\_t XIMC\_API** get\_uart\_settings ( **device\_t** id, **uart\_settings\_t** \* uart\_settings )

Read UART settings.

This function fill structure with UART settings.

See Also

[uart\\_settings\\_t](#)

Parameters

	<i>Speed</i>	UART speed
out	<i>uart_settings</i>	UART settings

7.1.4.91 **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.

7.1.4.92 **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

7.1.4.93 **result\_t XIMC\_API** load\_correction\_table ( **device\_t** \* id, const char \* namefile )

Command of loading a correction table from a text file (this function is deprecated).

Use the function [set\\_correction\\_table\(device\\_t id, const char\\* namefile\)](#). 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 fully qualified. If the short name is used, the file must be located in the application directory. If the file name is set to NULL, the correction table will be cleared. File format: two tab-separated columns. Column headers are string. Data is real, the point is a determiner. 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.

Note

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

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](#)

7.1.4.94 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

7.1.4.95 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

7.1.4.96 void **XIMC\_API** msec\_sleep ( unsigned int msec )

Sleeps for a specified amount of time.

Parameters

<i>msec</i>	time in milliseconds
-------------	----------------------

7.1.4.97 **device\_t** **XIMC\_API** open\_device ( const char \* uri )

Open a device with OS uri *uri* and return identifier of the device which can be used in calls.

## Parameters

<b>in</b>	<i>uri</i>	- a device uri. Device uri has form "xi-com:port" or "xi-net://host/serial" or "xi-emu:///file". In case of USB-COM port the "port" is the OS device uri. For example "xi-com:\\\\.COM3" in Windows or "xi-com:/dev/tty.s123" in Linux/Mac. In case of network device the "host" is an IPv4 address or fully qualified domain uri (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". Note: to open network device you must call <a href="#">set_bindy_key</a> first. In case of virtual device the "file" is the full filename with device memory state, if it doesn't exist then it is initialized with default values. For example "xi-emu:///C:/dir/file.bin" in Windows or "xi-emu:///home/user/file.bin" in Linux/-Mac.
-----------	------------	---

7.1.4.98 **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

<b>in</b>	<i>uri</i>	- a device uri
-----------	------------	----------------

7.1.4.99 **result\_t XIMC\_API** reset\_locks ( )

Resets the error of incorrect data transmission.

This function returns only 0 (OK). For example, sending the libximc command ends with an incorrect data transfer (error), any subsequent command always returns -1 (relevant for Windows).

7.1.4.100 **result\_t XIMC\_API** service\_command\_updf ( **device\_t** id )

Command puts the controller to update the firmware.

After receiving this command, the firmware board sets a flag (for loader), sends echo reply and restarts the controller.

7.1.4.101 **result\_t XIMC\_API** set\_accessories\_settings ( **device\_t** id, const **accessories\_settings\_t** \* accessories\_settings )

Set additional accessories information to EEPROM.

Can be used by manufacturer only.

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>accessories_settings</i>	structure contains information about additional accessories

7.1.4.102 **result\_t XIMC\_API** set\_bindy\_key ( const char \* keyfilepath )

Set network encryption layer (bindy) key.

Parameters

in	<i>keyfilepath</i>	full path to the bindy keyfile When using network-attached devices this function must be called before <a href="#">enumerate_devices</a> and <a href="#">open_device</a> functions.
----	--------------------	---

7.1.4.103 **result\_t XIMC\_API** set\_brake\_settings ( device\_t id, const brake\_settings\_t \* brake\_settings )

Set settings of brake control.

Parameters

	<i>id</i>	an identifier of device
in	<i>brake_settings</i>	structure contains settings of brake control

7.1.4.104 **result\_t XIMC\_API** set\_calibration\_settings ( device\_t id, const calibration\_settings\_t \* calibration\_settings )

Set calibration settings.

This function send structure with calibration settings to controller's memory.

See Also

[calibration\\_settings\\_t](#)

Parameters

	<i>id</i>	an identifier of device
in	<i>calibration_settings</i>	calibration settings

7.1.4.105 **result\_t XIMC\_API** set\_control\_settings ( device\_t id, const control\_settings\_t \* control\_settings )

Set settings of motor control.

When choosing CTL\_MODE = 1 switches motor control with the joystick. In this mode, the joystick to the maximum engine tends Move at MaxSpeed [i], where i = 0 if the previous use This mode is not selected another i. Buttons switch the room rate i. When CTL\_MODE = 2 is switched on motor control using the Left / right. When you click on the button motor starts to move in the appropriate direction at a speed MaxSpeed [0], at the end of time Timeout [i] motor move at a speed MaxSpeed [i+1]. at Transition from MaxSpeed [i] on MaxSpeed [i +1] to acceleration, as usual.

Parameters

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

7.1.4.106 **result\_t XIMC\_API** set\_control\_settings\_calb ( **device\_t** id, const **control\_settings\_calb\_t** \* control\_settings\_calb, const **calibration\_t** \* calibration )

Set settings of motor control which use user units.

When choosing CTL\_MODE = 1 switches motor control with the joystick. In this mode, the joystick to the maximum engine tends Move at MaxSpeed [i], where i = 0 if the previous use This mode is not selected another i. Buttons switch the room rate i. When CTL\_MODE = 2 is switched on motor control using the Left / right. When you click on the button motor starts to move in the appropriate direction at a speed MaxSpeed [0], at the end of time Timeout [i] motor move at a speed MaxSpeed [i+1]. at Transition from MaxSpeed [i] on MaxSpeed [i +1] to acceleration, as usual.

Parameters

	<i>id</i>	an identifier of device
<i>in</i>	<i>control_settings_calb</i>	structure contains settings motor control by joystick or buttons left/right.
	<i>calibration</i>	user unit settings

7.1.4.107 **result\_t XIMC\_API** set\_controller\_name ( **device\_t** id, const **controller\_name\_t** \* controller\_name )

Write user controller name and flags of setting from FRAM.

Parameters

	<i>id</i>	an identifier of device
<i>in</i>	<i>controller_name</i>	structure contains previously set user controller name

7.1.4.108 **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
<i>in</i>	<i>namefile</i>	- the file name must be fully qualified. If the short name is used, the file must be located in the application directory. If the file name is set to NULL, the correction table will be cleared. File format: two tab-separated columns. Column headers are string. Data is real, the point is a determiner. 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.

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](#)

7.1.4.109 **result\_t XIMC\_API** set\_ctp\_settings ( **device\_t** id, const **ctp\_settings\_t** \* ctp\_settings )

Set settings of control position(is only used with stepper motor).

When controlling the step motor with 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 (flag CTP\_ENABLED), the controller stores the current position in the footsteps of SM and the current position of the encoder. Further, at each step of the position encoder is converted into steps and if the difference is greater CTPMinError, a flag STATE\_CTP\_ERROR. When controlling the step motor with speed sensor (CTP\_BASE 1), the position is controlled by him. The active edge of input clock controller stores the current value of steps. Further, at each turn checks how many steps shifted. When a mismatch CTPMinError a flag STATE\_CTP\_ERROR.

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>ctp_settings</i>	structure contains settings of control position

7.1.4.110 **result\_t XIMC\_API** set\_debug\_write ( **device\_t** id, const **debug\_write\_t** \* debug\_write )

Write data to firmware for debug purpose.

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>debug_write</i>	Debug data.

7.1.4.111 **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 device
<b>in</b>	<i>edges_settings</i>	edges settings, specify types of borders, motor behaviour and electrical behaviour of limit switches

7.1.4.112 **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 which use user units.

See Also

[get\\_edges\\_settings\\_calb](#)

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>edges_settings_calb</i>	edges settings, specify types of borders, motor behaviour and electrical behaviour 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.

7.1.4.113 **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 download the new settings when you change the motor.

See Also

[get\\_emf\\_settings](#)

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>emf_settings</i>	EMF settings

7.1.4.114 **result\_t XIMC\_API** `set_encoder_information ( device_t id, const encoder_information_t * encoder_information )`

Set encoder information to EEPROM.

Can be used by manufacturer only.

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>encoder_information</i>	structure contains information about encoder

7.1.4.115 **result\_t XIMC\_API** `set_encoder_settings ( device_t id, const encoder_settings_t * encoder_settings )`

Set encoder settings to EEPROM.

Can be used by manufacturer only.

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>encoder_settings</i>	structure contains encoder settings

7.1.4.116 **result\_t XIMC\_API** set\_engine\_advanded\_setup ( **device\_t** id, const **engine\_advanded\_setup\_t** \* engine\_advanded\_setup )

Set engine advanded settings.

See Also

[get\\_engine\\_advanded\\_setup](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>engine- advanded_setup</i>	EAS settings

7.1.4.117 **result\_t XIMC\_API** set\_engine\_settings ( **device\_t** id, const **engine\_settings\_t** \* engine\_settings )

Set engine settings.

This function send structure with set of engine settings to controller's memory. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics. Use it when you change motor, encoder, positioner etc. Please note that wrong engine settings lead to device malfunction, can lead to irreversible damage of board.

See Also

[get\\_engine\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>engine_settings</i>	engine settings

7.1.4.118 **result\_t XIMC\_API** set\_engine\_settings\_calb ( **device\_t** id, const **engine\_settings\_calb\_t** \* engine\_settings\_calb, const **calibration\_t** \* calibration )

Set engine settings which use user units.

This function send structure with set of engine settings to controller's memory. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics. Use it when you change motor, encoder, positioner etc. Please note that wrong engine settings lead to device malfunction, can lead to irreversible damage of board.

See Also

[get\\_engine\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>engine_settings- _calb</i>	engine settings
	<i>calibration</i>	user unit settings

7.1.4.119 **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 device
<b>in</b>	<i>entype_settings</i>	structure contains settings motor type and power driver type

7.1.4.120 **result\_t XIMC\_API** set\_extended\_settings ( **device\_t** id, const **extended\_settings\_t** \* extended\_settings )

Set extended settings.

See Also

[get\\_extended\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>extended_settings</i>	EST settings

7.1.4.121 **result\_t XIMC\_API** set\_extio\_settings ( **device\_t** id, const **extio\_settings\_t** \* extio\_settings )

Set EXTIO settings.

This function writes a structure with a set of EXTIO settings to controller's memory. By default input event are signalled through rising front and output states are signalled by high logic state.

See Also

[get\\_extio\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>extio_settings</i>	EXTIO settings

7.1.4.122 **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 device
<b>in</b>	<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.

<code>in</code>	<i>FeedbackType</i>	type of feedback
<code>in</code>	<i>FeedbackFlags</i>	flags of feedback
<code>in</code>	<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.

7.1.4.123 **result\_t XIMC\_API** set\_gear\_information ( **device\_t** id, const **gear\_information\_t** \* gear\_information )

Set gear information to EEPROM.

Can be used by manufacturer only.

Parameters

	<i>id</i>	an identifier of device
<code>in</code>	<i>gear_information</i>	structure contains information about step gearhead

7.1.4.124 **result\_t XIMC\_API** set\_gear\_settings ( **device\_t** id, const **gear\_settings\_t** \* gear\_settings )

Set gear settings to EEPROM.

Can be used by manufacturer only.

Parameters

	<i>id</i>	an identifier of device
<code>in</code>	<i>gear_settings</i>	structure contains step gearhead settings

7.1.4.125 **result\_t XIMC\_API** set\_hallsensor\_information ( **device\_t** id, const **hallsensor\_information\_t** \* hallsensor\_information )

Set hall sensor information to EEPROM.

Can be used by manufacturer only.

Parameters

	<i>id</i>	an identifier of device
<code>in</code>	<i>hallsensor_information</i>	structure contains information about hall sensor

7.1.4.126 **result\_t XIMC\_API** set\_hallsensor\_settings ( **device\_t** id, const **hallsensor\_settings\_t** \* hallsensor\_settings )

Set hall sensor settings to EEPROM.

Can be used by manufacturer only.

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>hallsensor_settings</i>	structure contains hall sensor settings

7.1.4.127 **result\_t XIMC\_API** set\_home\_settings ( **device\_t** id, const **home\_settings\_t** \* home\_settings )

Set home settings.

This function send structure with calibrating position settings to controller's memory.

See Also

[home\\_settings\\_t](#)

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>home_settings</i>	calibrating position settings

7.1.4.128 **result\_t XIMC\_API** set\_home\_settings\_calb ( **device\_t** id, const **home\_settings\_calb\_t** \* home\_settings\_calb, const **calibration\_t** \* calibration )

Set home settings which use user units.

This function send structure with calibrating position settings to controller's memory.

See Also


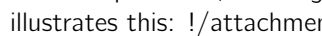
[home\\_settings\\_calb\\_t](#)

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>home_settings_calb</i>	calibrating position settings
	<i>calibration</i>	user unit settings

7.1.4.129 **result\_t XIMC\_API** set\_joystick\_settings ( **device\_t** id, const **joystick\_settings\_t** \* joystick\_settings )

Set settings of joystick.

If joystick position is outside DeadZone limits from the central position a movement with speed, defined by the joystick DeadZone edge to 100% deviation, begins. Joystick positions inside DeadZone limits correspond to zero speed (soft stop of motion) and positions beyond Low and High limits correspond MaxSpeed [i] or -MaxSpeed [i] (see command SCTL), where i = 0 by default and can be changed with left/right buttons (see command SCTL). If next speed in list is zero (both integer and microstep parts), the button press is ignored. First speed in list shouldn't be zero. The DeadZone ranges are illustrated on the following picture.  The relationship between the deviation and the rate is exponential, allowing no switching speed combine high mobility and accuracy. The following picture illustrates this:  The nonlinearity parameter is adjustable. Setting it to zero makes deviation/speed relation linear.

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>joystick_settings</i>	structure contains joystick settings

7.1.4.130 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
-------------------------	-----------------------------

7.1.4.131 **result\_t XIMC\_API** set\_motor\_information ( **device\_t** id, const **motor\_information\_t** \* motor\_information )

Set motor information to EEPROM.

Can be used by manufacturer only.

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>motor_information</i>	structure contains motor information

7.1.4.132 **result\_t XIMC\_API** set\_motor\_settings ( **device\_t** id, const **motor\_settings\_t** \* motor\_settings )

Set motor settings to EEPROM.

Can be used by manufacturer only.

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>motor_settings</i>	structure contains motor information

7.1.4.133 **result\_t XIMC\_API** set\_move\_settings ( **device\_t** id, const **move\_settings\_t** \* move\_settings )

Set command setup movement (speed, acceleration, threshold and etc).

## Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>move_settings</i>	structure contains move settings: speed, acceleration, deceleration etc.

7.1.4.134 **result\_t XIMC\_API** set\_move\_settings\_calb ( **device\_t** id, const **move\_settings\_calb\_t** \* move\_settings\_calb, const **calibration\_t** \* calibration )

Set command setup movement which use user units (speed, acceleration, threshold and etc).

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>move_settings_calb</i>	structure contains move settings: speed, acceleration, deceleration etc.
	<i>calibration</i>	user unit settings

7.1.4.135 **result\_t XIMC\_API** set\_nonvolatile\_memory ( **device\_t** id, const **nonvolatile\_memory\_t** \* nonvolatile\_memory )

Write userdata into FRAM.

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>nonvolatile_memory</i>	structure contains previously set userdata

7.1.4.136 **result\_t XIMC\_API** set\_pid\_settings ( **device\_t** id, const **pid\_settings\_t** \* pid\_settings )

Set PID settings.

This function send structure with set of PID factors to controller's memory. These settings specify behaviour of PID routine for positioner. These factors are slightly different for different positioners. All boards are supplied with standard set of PID setting on 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 device
<b>in</b>	<i>pid_settings</i>	pid settings

7.1.4.137 **result\_t XIMC\_API** set\_position ( **device\_t** id, const **set\_position\_t** \* the\_set\_position )

Sets any position value in steps and micro for stepper motor and encoder steps of all engines.

It means, that changing main indicator of position.

Parameters

	<i>id</i>	an identifier of device
<b>out</b>	<i>the_set_position</i>	structure contains move settings: speed, acceleration, deceleration etc.

7.1.4.138 **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 which use user units.

It means, that changing main indicator of position.

Parameters

	<i>id</i>	an identifier of device
out	<i>the_set_position_calb</i>	structure contains move settings: speed, acceleration, deceleration etc.
	<i>calibration</i>	user unit settings

7.1.4.139 **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 stepper motor only.

Parameters

	<i>id</i>	an identifier of device
in	<i>power_settings</i>	structure contains settings of step motor power control

7.1.4.140 **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 device
	<i>secure_settings</i>	structure with secure data

See Also

status\_t::flags

7.1.4.141 **result\_t XIMC\_API** set\_serial\_number ( **device\_t** id, const **serial\_number\_t** \* serial\_number )

Write device serial number and hardware version to 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 manufacturer only.

Parameters

	<i>id</i>	an identifier of device
in	<i>serial_number</i>	structure contains new serial number and secret key.

7.1.4.142 **result\_t XIMC\_API** set\_stage\_information ( **device\_t** id, const **stage\_information\_t** \* stage\_information )

Set stage information to EEPROM.

Can be used by manufacturer only.

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>stage-information</i>	structure contains stage information

7.1.4.143 **result\_t XIMC\_API** set\_stage\_name ( **device\_t** id, const **stage\_name\_t** \* stage\_name )

Write user stage name from EEPROM.

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>stage_name</i>	structure contains previously set user stage name

7.1.4.144 **result\_t XIMC\_API** set\_stage\_settings ( **device\_t** id, const **stage\_settings\_t** \* stage\_settings )

Set stage settings to EEPROM.

Can be used by manufacturer only

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>stage_settings</i>	structure contains stage settings

7.1.4.145 **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 send structure with set of input synchronization settings, that specify behaviour of input synchronization, to controller's memory. All boards are supplied with standard set of these settings.

See Also

[get\\_sync\\_in\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>sync_in_settings</i>	synchronization settings

7.1.4.146 **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 synchronization settings which use user units.

This function send structure with set of input synchronization settings, that specify behaviour of input synchronization, to controller's memory. All boards are supplied with standard set of these settings.

See Also

[get\\_sync\\_in\\_settings\\_calb](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>sync_in_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

7.1.4.147 **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 send structure with set of output synchronization settings, that specify behaviour of output synchronization, to controller's memory. All boards are supplied with standard set of these settings.

See Also

[get\\_sync\\_out\\_settings](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>sync_out_settings</i>	synchronization settings

7.1.4.148 **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 synchronization settings which use user units.

This function send structure with set of output synchronization settings, that specify behaviour of output synchronization, to controller's memory. All boards are supplied with standard set of these settings.

See Also

[get\\_sync\\_in\\_settings\\_calb](#)

Parameters

	<i>id</i>	an identifier of device
<b>in</b>	<i>sync_out_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

7.1.4.149 **result\_t XIMC\_API** set\_uart\_settings ( **device\_t** id, const **uart\_settings\_t** \* uart\_settings )

Set UART settings.

This function send structure with UART settings to controller's memory.

See Also

[uart\\_settings\\_t](#)

Parameters

	<i>Speed</i>	UART speed
<i>in</i>	<i>uart_settings</i>	UART settings

7.1.4.150 **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
<i>in</i>	<i>key</i>	protection key. Range: 0..4294967295

7.1.4.151 **result\_t XIMC\_API** ximc\_fix\_usbser\_sys ( const char \* device\_uri )

Fixing a USB driver error in Windows.

The USB-COM subsystem in the Windows OS does not always work correctly. During operation, the following malfunctions are possible: All attempts to open the device fail. The device can be opened and data can be sent to it, but the response data is not received. These problems are fixed by reconnecting the device or reinitializing it in the Device Manager. The [ximc\\_fix\\_usbser\\_sys\(\)](#) function automates the deletion detection process.

7.1.4.152 **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
----------------	---

# Index

A1Voltage  
  analog\_data\_t, [17](#)  
A1Voltage\_ADC  
  analog\_data\_t, [17](#)  
A2Voltage  
  analog\_data\_t, [17](#)  
A2Voltage\_ADC  
  analog\_data\_t, [17](#)  
ACurrent  
  analog\_data\_t, [18](#)  
ACurrent\_ADC  
  analog\_data\_t, [18](#)  
Accel  
  move\_settings\_calb\_t, [58](#)  
  move\_settings\_t, [59](#)  
accessories\_settings\_t, [14](#)  
  LimitSwitchesSettings, [15](#)  
  MBRatedCurrent, [15](#)  
  MBRatedVoltage, [15](#)  
  MBSettings, [15](#)  
  MBTorque, [15](#)  
  MagneticBrakeInfo, [15](#)  
  TSGrad, [15](#)  
  TSMaх, [15](#)  
  TSMin, [15](#)  
  TSSettings, [16](#)  
  TemperatureSensorInfo, [15](#)  
Accuracy  
  sync\_out\_settings\_calb\_t, [77](#)  
  sync\_out\_settings\_t, [78](#)  
analog\_data\_t, [16](#)  
  A1Voltage, [17](#)  
  A1Voltage\_ADC, [17](#)  
  A2Voltage, [17](#)  
  A2Voltage\_ADC, [17](#)  
  ACurrent, [18](#)  
  ACurrent\_ADC, [18](#)  
  B1Voltage, [18](#)  
  B1Voltage\_ADC, [18](#)  
  B2Voltage, [18](#)  
  B2Voltage\_ADC, [18](#)  
  BCurrent, [18](#)  
  BCurrent\_ADC, [18](#)  
  FullCurrent, [18](#)  
  FullCurrent\_ADC, [18](#)  
  H5, [18](#)  
  Joy, [18](#)  
  Joy\_ADC, [19](#)  
  L, [19](#)  
  L5, [19](#)  
  L5\_ADC, [19](#)  
  Pot, [19](#)  
  R, [19](#)  
  SupVoltage, [19](#)  
  SupVoltage\_ADC, [19](#)  
  Temp, [19](#)  
  Temp\_ADC, [19](#)  
Antiplay  
  engine\_settings\_calb\_t, [37](#)  
  engine\_settings\_t, [38](#)  
AntiplaySpeed  
  move\_settings\_calb\_t, [58](#)  
  move\_settings\_t, [59](#)  
  
B1Voltage  
  analog\_data\_t, [18](#)  
B1Voltage\_ADC  
  analog\_data\_t, [18](#)  
B2Voltage  
  analog\_data\_t, [18](#)  
B2Voltage\_ADC  
  analog\_data\_t, [18](#)  
BACK\_EMF\_KM\_AUTO  
  ximc.h, [104](#)  
BCurrent  
  analog\_data\_t, [18](#)  
BCurrent\_ADC  
  analog\_data\_t, [18](#)  
BORDER\_IS\_ENCODER  
  ximc.h, [104](#)  
BORDER\_STOP\_LEFT  
  ximc.h, [105](#)  
BORDER\_STOP\_RIGHT  
  ximc.h, [105](#)  
BRAKE\_ENABLED  
  ximc.h, [105](#)  
BRAKE\_ENG\_PWROFF  
  ximc.h, [105](#)  
BackEMFFlags  
  emf\_settings\_t, [33](#)  
BorderFlags  
  edges\_settings\_calb\_t, [31](#)  
  edges\_settings\_t, [32](#)  
brake\_settings\_t, [20](#)  
  BrakeFlags, [20](#)

- t1, [20](#)
- t2, [20](#)
- t3, [20](#)
- t4, [20](#)
- BrakeFlags
  - brake\_settings\_t, [20](#)
- CONTROL\_MODE\_BITS
  - ximc.h, [105](#)
- CONTROL\_MODE\_JOY
  - ximc.h, [105](#)
- CONTROL\_MODE\_LR
  - ximc.h, [105](#)
- CONTROL\_MODE\_OFF
  - ximc.h, [105](#)
- CSS1\_A
  - calibration\_settings\_t, [21](#)
- CSS1\_B
  - calibration\_settings\_t, [21](#)
- CSS2\_A
  - calibration\_settings\_t, [21](#)
- CSS2\_B
  - calibration\_settings\_t, [21](#)
- CTP\_ALARM\_ON\_ERROR
  - ximc.h, [105](#)
- CTP\_BASE
  - ximc.h, [106](#)
- CTP\_ENABLED
  - ximc.h, [106](#)
- CTP\_ERROR\_CORRECTION
  - ximc.h, [106](#)
- CTPFlags
  - ctp\_settings\_t, [28](#)
- CTPMinError
  - ctp\_settings\_t, [28](#)
- calibration\_settings\_t, [21](#)
  - CSS1\_A, [21](#)
  - CSS1\_B, [21](#)
  - CSS2\_A, [21](#)
  - CSS2\_B, [21](#)
  - FullCurrent\_A, [21](#)
  - FullCurrent\_B, [22](#)
- calibration\_t, [22](#)
- chart\_data\_t, [22](#)
  - DutyCycle, [23](#)
  - Joy, [23](#)
  - Pot, [23](#)
  - WindingCurrentA, [23](#)
  - WindingCurrentB, [23](#)
  - WindingCurrentC, [23](#)
  - WindingVoltageA, [23](#)
  - WindingVoltageB, [24](#)
  - WindingVoltageC, [24](#)
- close\_device
  - ximc.h, [120](#)
- ClutterTime
  - sync\_in\_settings\_calb\_t, [75](#)
  - sync\_in\_settings\_t, [76](#)
- CmdBufFreeSpace
  - status\_calb\_t, [70](#)
  - status\_t, [73](#)
- command\_clear\_fram
  - ximc.h, [120](#)
- command\_eeread\_settings
  - ximc.h, [120](#)
- command\_eesave\_settings
  - ximc.h, [120](#)
- command\_home
  - ximc.h, [121](#)
- command\_homezero
  - ximc.h, [121](#)
- command\_left
  - ximc.h, [121](#)
- command\_loft
  - ximc.h, [121](#)
- command\_move
  - ximc.h, [122](#)
- command\_move\_calb
  - ximc.h, [122](#)
- command\_movr
  - ximc.h, [122](#)
- command\_movr\_calb
  - ximc.h, [123](#)
- command\_power\_off
  - ximc.h, [123](#)
- command\_read\_robust\_settings
  - ximc.h, [123](#)
- command\_read\_settings
  - ximc.h, [123](#)
- command\_reset
  - ximc.h, [124](#)
- command\_right
  - ximc.h, [124](#)
- command\_save\_robust\_settings
  - ximc.h, [124](#)
- command\_save\_settings
  - ximc.h, [124](#)
- command\_sstp
  - ximc.h, [124](#)
- command\_start\_measurements
  - ximc.h, [125](#)
- command\_stop
  - ximc.h, [125](#)
- command\_update\_firmware
  - ximc.h, [125](#)
- command\_wait\_for\_stop
  - ximc.h, [125](#)
- command\_zero
  - ximc.h, [125](#)
- control\_settings\_calb\_t, [24](#)
  - Flags, [25](#)
  - MaxClickTime, [25](#)

- MaxSpeed, [25](#)
- Timeout, [25](#)
- control\_settings\_t, [25](#)
  - Flags, [26](#)
  - MaxClickTime, [26](#)
  - MaxSpeed, [26](#)
  - Timeout, [26](#)
  - uDeltaPosition, [26](#)
  - uMaxSpeed, [26](#)
- controller\_name\_t, [26](#)
  - ControllerName, [27](#)
  - CtrlFlags, [27](#)
- ControllerName
  - controller\_name\_t, [27](#)
- CountsPerTurn
  - feedback\_settings\_t, [42](#)
- CriticalIpwr
  - secure\_settings\_t, [63](#)
- CriticalIusb
  - secure\_settings\_t, [63](#)
- CriticalT
  - secure\_settings\_t, [63](#)
- CriticalUpwr
  - secure\_settings\_t, [63](#)
- CriticalUusb
  - secure\_settings\_t, [63](#)
- ctp\_settings\_t, [27](#)
  - CTPFlags, [28](#)
  - CTPMinError, [28](#)
- CtrlFlags
  - controller\_name\_t, [27](#)
- CurPosition
  - status\_calb\_t, [70](#)
  - status\_t, [73](#)
- CurSpeed
  - status\_calb\_t, [71](#)
  - status\_t, [73](#)
- CurT
  - status\_calb\_t, [71](#)
  - status\_t, [73](#)
- CurrReductDelay
  - power\_settings\_t, [62](#)
- CurrentSetTime
  - power\_settings\_t, [62](#)
- DRIVER\_TYPE\_EXTERNAL
  - ximc.h, [106](#)
- DeadZone
  - joystick\_settings\_t, [52](#)
- debug\_read\_t, [28](#)
  - DebugData, [28](#)
- debug\_write\_t, [28](#)
  - DebugData, [29](#)
- DebugData
  - debug\_read\_t, [28](#)
  - debug\_write\_t, [29](#)
- Decel
  - move\_settings\_calb\_t, [58](#)
  - move\_settings\_t, [60](#)
- DetentTorque
  - motor\_settings\_t, [55](#)
- device\_information\_t, [29](#)
  - Major, [29](#)
  - Minor, [29](#)
  - Release, [30](#)
- device\_network\_information\_t, [30](#)
- DriverType
  - entype\_settings\_t, [40](#)
- DutyCycle
  - chart\_data\_t, [23](#)
- EEPROM\_PRECEDENCE
  - ximc.h, [106](#)
- ENC\_STATE\_ABSENT
  - ximc.h, [106](#)
- ENC\_STATE\_MALFUNC
  - ximc.h, [106](#)
- ENC\_STATE\_OK
  - ximc.h, [106](#)
- ENC\_STATE\_REVERS
  - ximc.h, [106](#)
- ENC\_STATE\_UNKNOWN
  - ximc.h, [107](#)
- ENDER\_SW1\_ACTIVE\_LOW
  - ximc.h, [107](#)
- ENDER\_SW2\_ACTIVE\_LOW
  - ximc.h, [107](#)
- ENDER\_SWAP
  - ximc.h, [107](#)
- ENGINE\_ACCEL\_ON
  - ximc.h, [107](#)
- ENGINE\_ANTIPLAY
  - ximc.h, [107](#)
- ENGINE\_LIMIT\_CURR
  - ximc.h, [107](#)
- ENGINE\_LIMIT\_RPM
  - ximc.h, [107](#)
- ENGINE\_LIMIT\_VOLT
  - ximc.h, [107](#)
- ENGINE\_MAX\_SPEED
  - ximc.h, [108](#)
- ENGINE\_REVERSE
  - ximc.h, [108](#)
- ENGINE\_TYPE\_2DC
  - ximc.h, [108](#)
- ENGINE\_TYPE\_DC
  - ximc.h, [108](#)
- ENGINE\_TYPE\_NONE
  - ximc.h, [108](#)
- ENGINE\_TYPE\_STEP
  - ximc.h, [108](#)
- ENGINE\_TYPE\_TEST

- ximc.h, [108](#)
- ENUMERATE\_PROBE
  - ximc.h, [108](#)
- EXTIO\_SETUP\_INVERT
  - ximc.h, [108](#)
- EXTIO\_SETUP\_OUTPUT
  - ximc.h, [110](#)
- EXTIOModeFlags
  - extio\_settings.t, [41](#)
- EXTIOSetupFlags
  - extio\_settings.t, [41](#)
- edges\_settings\_calb.t, [30](#)
  - BorderFlags, [31](#)
  - EnderFlags, [31](#)
  - LeftBorder, [31](#)
  - RightBorder, [31](#)
- edges\_settings.t, [31](#)
  - BorderFlags, [32](#)
  - EnderFlags, [32](#)
  - LeftBorder, [32](#)
  - RightBorder, [32](#)
  - uLeftBorder, [32](#)
  - uRightBorder, [32](#)
- Efficiency
  - gear\_settings.t, [43](#)
- emf\_settings.t, [32](#)
  - BackEMFFlags, [33](#)
  - Km, [33](#)
  - L, [33](#)
  - R, [33](#)
- EncPosition
  - get\_position\_calb.t, [45](#)
  - get\_position.t, [45](#)
  - set\_position\_calb.t, [65](#)
  - set\_position.t, [66](#)
  - status\_calb.t, [71](#)
  - status.t, [73](#)
- EncSts
  - status\_calb.t, [71](#)
  - status.t, [73](#)
- encoder\_information.t, [33](#)
  - Manufacturer, [34](#)
  - PartNumber, [34](#)
- encoder\_settings.t, [34](#)
  - EncoderSettings, [35](#)
  - MaxCurrentConsumption, [35](#)
  - MaxOperatingFrequency, [35](#)
  - SupplyVoltageMax, [35](#)
  - SupplyVoltageMin, [35](#)
- EncoderSettings
  - encoder\_settings.t, [35](#)
- EnderFlags
  - edges\_settings\_calb.t, [31](#)
  - edges\_settings.t, [32](#)
- engine\_advanced\_setup.t, [35](#)
  - stepcloseloop\_Kp\_high, [36](#)
  - stepcloseloop\_Kp\_low, [36](#)
  - stepcloseloop\_Kw, [36](#)
- engine\_settings\_calb.t, [36](#)
  - Antiplay, [37](#)
  - EngineFlags, [37](#)
  - MicrostepMode, [37](#)
  - NomCurrent, [37](#)
  - NomSpeed, [37](#)
  - NomVoltage, [37](#)
  - StepsPerRev, [37](#)
- engine\_settings.t, [38](#)
  - Antiplay, [38](#)
  - EngineFlags, [38](#)
  - MicrostepMode, [38](#)
  - NomCurrent, [39](#)
  - NomSpeed, [39](#)
  - NomVoltage, [39](#)
  - StepsPerRev, [39](#)
  - uNomSpeed, [39](#)
- EngineFlags
  - engine\_settings\_calb.t, [37](#)
  - engine\_settings.t, [38](#)
- EngineType
  - entype\_settings.t, [40](#)
- entype\_settings.t, [39](#)
  - DriverType, [40](#)
  - EngineType, [40](#)
- enumerate\_devices
  - ximc.h, [126](#)
- Error
  - measurements.t, [53](#)
- ExpFactor
  - joystick\_settings.t, [52](#)
- extended\_settings.t, [40](#)
- extio\_settings.t, [40](#)
  - EXTIOModeFlags, [41](#)
  - EXTIOSetupFlags, [41](#)
- FEEDBACK\_EMF
  - ximc.h, [110](#)
- FEEDBACK\_ENC\_REVERSE
  - ximc.h, [110](#)
- FEEDBACK\_ENCODER
  - ximc.h, [110](#)
- FEEDBACK\_NONE
  - ximc.h, [110](#)
- FastHome
  - home\_settings\_calb.t, [49](#)
  - home\_settings.t, [50](#)
- feedback\_settings.t, [41](#)
  - CountsPerTurn, [42](#)
  - FeedbackFlags, [42](#)
  - FeedbackType, [42](#)
  - IPS, [42](#)
- FeedbackFlags
  - feedback\_settings.t, [42](#)

- FeedbackType
  - feedback\_settings\_t, [42](#)
- Flags
  - control\_settings\_calb\_t, [25](#)
  - control\_settings\_t, [26](#)
  - secure\_settings\_t, [64](#)
  - status\_calb\_t, [71](#)
  - status\_t, [73](#)
- free\_enumerate\_devices
  - ximc.h, [126](#)
- FullCurrent
  - analog\_data\_t, [18](#)
- FullCurrent\_A
  - calibration\_settings\_t, [21](#)
- FullCurrent\_ADC
  - analog\_data\_t, [18](#)
- FullCurrent\_B
  - calibration\_settings\_t, [22](#)
- GPIOFlags
  - status\_calb\_t, [71](#)
  - status\_t, [73](#)
- gear\_information\_t, [42](#)
  - Manufacturer, [43](#)
  - PartNumber, [43](#)
- gear\_settings\_t, [43](#)
  - Efficiency, [43](#)
  - InputInertia, [43](#)
  - MaxOutputBacklash, [44](#)
  - RatedInputSpeed, [44](#)
  - RatedInputTorque, [44](#)
  - ReductionIn, [44](#)
  - ReductionOut, [44](#)
- get\_accessories\_settings
  - ximc.h, [126](#)
- get\_analog\_data
  - ximc.h, [126](#)
- get\_bootloader\_version
  - ximc.h, [127](#)
- get\_brake\_settings
  - ximc.h, [127](#)
- get\_calibration\_settings
  - ximc.h, [127](#)
- get\_chart\_data
  - ximc.h, [127](#)
- get\_control\_settings
  - ximc.h, [128](#)
- get\_control\_settings\_calb
  - ximc.h, [128](#)
- get\_controller\_name
  - ximc.h, [128](#)
- get\_ctp\_settings
  - ximc.h, [129](#)
- get\_debug\_read
  - ximc.h, [129](#)
- get\_device\_count
  - ximc.h, [129](#)
- get\_device\_information
  - ximc.h, [129](#)
- get\_device\_name
  - ximc.h, [130](#)
- get\_edges\_settings
  - ximc.h, [130](#)
- get\_edges\_settings\_calb
  - ximc.h, [130](#)
- get\_emf\_settings
  - ximc.h, [130](#)
- get\_encoder\_information
  - ximc.h, [131](#)
- get\_encoder\_settings
  - ximc.h, [131](#)
- get\_engine\_advanced\_setup
  - ximc.h, [131](#)
- get\_engine\_settings
  - ximc.h, [131](#)
- get\_engine\_settings\_calb
  - ximc.h, [132](#)
- get\_entype\_settings
  - ximc.h, [132](#)
- get\_enumerate\_device\_controller\_name
  - ximc.h, [132](#)
- get\_enumerate\_device\_information
  - ximc.h, [133](#)
- get\_enumerate\_device\_network\_information
  - ximc.h, [133](#)
- get\_enumerate\_device\_serial
  - ximc.h, [133](#)
- get\_enumerate\_device\_stage\_name
  - ximc.h, [133](#)
- get\_extended\_settings
  - ximc.h, [134](#)
- get\_extio\_settings
  - ximc.h, [134](#)
- get\_feedback\_settings
  - ximc.h, [134](#)
- get\_firmware\_version
  - ximc.h, [135](#)
- get\_gear\_information
  - ximc.h, [135](#)
- get\_gear\_settings
  - ximc.h, [135](#)
- get\_globally\_unique\_identifier
  - ximc.h, [135](#)
- get\_hallsensor\_information
  - ximc.h, [136](#)
- get\_hallsensor\_settings
  - ximc.h, [136](#)
- get\_home\_settings
  - ximc.h, [136](#)
- get\_home\_settings\_calb
  - ximc.h, [136](#)
- get\_init\_random

- ximc.h, [137](#)
- get\_joystick\_settings
  - ximc.h, [137](#)
- get\_measurements
  - ximc.h, [137](#)
- get\_motor\_information
  - ximc.h, [137](#)
- get\_motor\_settings
  - ximc.h, [138](#)
- get\_move\_settings
  - ximc.h, [138](#)
- get\_move\_settings\_calb
  - ximc.h, [138](#)
- get\_nonvolatile\_memory
  - ximc.h, [138](#)
- get\_pid\_settings
  - ximc.h, [138](#)
- get\_position
  - ximc.h, [139](#)
- get\_position\_calb
  - ximc.h, [139](#)
- get\_position\_calb\_t, [44](#)
  - EncPosition, [45](#)
  - Position, [45](#)
- get\_position\_t, [45](#)
  - EncPosition, [45](#)
  - uPosition, [45](#)
- get\_power\_settings
  - ximc.h, [139](#)
- get\_secure\_settings
  - ximc.h, [139](#)
- get\_serial\_number
  - ximc.h, [140](#)
- get\_stage\_information
  - ximc.h, [140](#)
- get\_stage\_name
  - ximc.h, [140](#)
- get\_stage\_settings
  - ximc.h, [140](#)
- get\_status
  - ximc.h, [140](#)
- get\_status\_calb
  - ximc.h, [141](#)
- get\_sync\_in\_settings
  - ximc.h, [141](#)
- get\_sync\_in\_settings\_calb
  - ximc.h, [141](#)
- get\_sync\_out\_settings
  - ximc.h, [142](#)
- get\_sync\_out\_settings\_calb
  - ximc.h, [142](#)
- get\_uart\_settings
  - ximc.h, [142](#)
- globally\_unique\_identifier\_t, [46](#)
  - UniqueID0, [46](#)
  - UniqueID1, [46](#)
  - UniqueID2, [46](#)
  - UniqueID3, [46](#)
- goto\_firmware
  - ximc.h, [143](#)
- H5
  - analog\_data\_t, [18](#)
- H\_BRIDGE\_ALERT
  - ximc.h, [110](#)
- HOME\_DIR\_FIRST
  - ximc.h, [111](#)
- HOME\_DIR\_SECOND
  - ximc.h, [111](#)
- HOME\_HALF\_MV
  - ximc.h, [111](#)
- HOME\_MV\_SEC\_EN
  - ximc.h, [111](#)
- HOME\_STOP\_FIRST\_LIM
  - ximc.h, [111](#)
- HOME\_STOP\_FIRST\_REV
  - ximc.h, [111](#)
- HOME\_STOP\_FIRST\_SYN
  - ximc.h, [111](#)
- HOME\_USE\_FAST
  - ximc.h, [112](#)
- hallsensor\_information\_t, [46](#)
  - Manufacturer, [47](#)
  - PartNumber, [47](#)
- hallsensor\_settings\_t, [47](#)
  - MaxCurrentConsumption, [48](#)
  - MaxOperatingFrequency, [48](#)
  - SupplyVoltageMax, [48](#)
  - SupplyVoltageMin, [48](#)
- has\_firmware
  - ximc.h, [143](#)
- HoldCurrent
  - power\_settings\_t, [62](#)
- home\_settings\_calb\_t, [48](#)
  - FastHome, [49](#)
  - HomeDelta, [49](#)
  - HomeFlags, [49](#)
  - SlowHome, [49](#)
- home\_settings\_t, [49](#)
  - FastHome, [50](#)
  - HomeDelta, [50](#)
  - HomeFlags, [50](#)
  - SlowHome, [50](#)
  - uFastHome, [50](#)
  - uHomeDelta, [50](#)
  - uSlowHome, [50](#)
- HomeDelta
  - home\_settings\_calb\_t, [49](#)
  - home\_settings\_t, [50](#)
- HomeFlags
  - home\_settings\_calb\_t, [49](#)
  - home\_settings\_t, [50](#)

- HorizontalLoadCapacity
  - stage\_settings\_t, [68](#)
- IPS
  - feedback\_settings\_t, [42](#)
- init\_random\_t, [51](#)
  - key, [51](#)
- InputInertia
  - gear\_settings\_t, [43](#)
- lpwr
  - status\_calb\_t, [71](#)
  - status\_t, [74](#)
- lusb
  - status\_calb\_t, [71](#)
  - status\_t, [74](#)
- JOY\_REVERSE
  - ximc.h, [112](#)
- Joy
  - analog\_data\_t, [18](#)
  - chart\_data\_t, [23](#)
- Joy\_ADC
  - analog\_data\_t, [19](#)
- JoyCenter
  - joystick\_settings\_t, [52](#)
- JoyFlags
  - joystick\_settings\_t, [52](#)
- JoyHighEnd
  - joystick\_settings\_t, [52](#)
- JoyLowEnd
  - joystick\_settings\_t, [52](#)
- joystick\_settings\_t, [51](#)
  - DeadZone, [52](#)
  - ExpFactor, [52](#)
  - JoyCenter, [52](#)
  - JoyFlags, [52](#)
  - JoyHighEnd, [52](#)
  - JoyLowEnd, [52](#)
- Key
  - serial\_number\_t, [64](#)
- key
  - init\_random\_t, [51](#)
- Km
  - emf\_settings\_t, [33](#)
- L
  - analog\_data\_t, [19](#)
  - emf\_settings\_t, [33](#)
- L5
  - analog\_data\_t, [19](#)
- L5\_ADC
  - analog\_data\_t, [19](#)
- LOW\_UPWR\_PROTECTION
  - ximc.h, [112](#)
- LeadScrewPitch
  - stage\_settings\_t, [68](#)
- LeftBorder
  - edges\_settings\_calb\_t, [31](#)
  - edges\_settings\_t, [32](#)
- Length
  - measurements\_t, [53](#)
- LimitSwitchesSettings
  - accessories\_settings\_t, [15](#)
- load\_correction\_table
  - ximc.h, [143](#)
- logging\_callback\_stderr\_narrow
  - ximc.h, [144](#)
- logging\_callback\_stderr\_wide
  - ximc.h, [144](#)
- logging\_callback\_t
  - ximc.h, [120](#)
- LowUpwrOff
  - secure\_settings\_t, [64](#)
- MBRatedCurrent
  - accessories\_settings\_t, [15](#)
- MBRatedVoltage
  - accessories\_settings\_t, [15](#)
- MBSettings
  - accessories\_settings\_t, [15](#)
- MBTorque
  - accessories\_settings\_t, [15](#)
- MICROSTEP\_MODE\_FULL
  - ximc.h, [113](#)
- MOVE\_STATE\_ANTIPLAY
  - ximc.h, [113](#)
- MOVE\_STATE\_MOVING
  - ximc.h, [113](#)
- MVCMD\_ERROR
  - ximc.h, [113](#)
- MVCMD\_HOME
  - ximc.h, [113](#)
- MVCMD\_LEFT
  - ximc.h, [113](#)
- MVCMD\_LOFT
  - ximc.h, [113](#)
- MVCMD\_MOVE
  - ximc.h, [113](#)
- MVCMD\_MOVR
  - ximc.h, [113](#)
- MVCMD\_NAME\_BITS
  - ximc.h, [113](#)
- MVCMD\_RIGHT
  - ximc.h, [114](#)
- MVCMD\_RUNNING
  - ximc.h, [114](#)
- MVCMD\_SSTP
  - ximc.h, [114](#)
- MVCMD\_STOP
  - ximc.h, [114](#)
- MVCMD\_UKNWN
  - ximc.h, [114](#)

- MagneticBrakeInfo
  - accessories\_settings\_t, [15](#)
- Major
  - device\_information\_t, [29](#)
  - serial\_number\_t, [64](#)
- Manufacturer
  - encoder\_information\_t, [34](#)
  - gear\_information\_t, [43](#)
  - hallsensor\_information\_t, [47](#)
  - motor\_information\_t, [54](#)
  - stage\_information\_t, [67](#)
- MaxClickTime
  - control\_settings\_calb\_t, [25](#)
  - control\_settings\_t, [26](#)
- MaxCurrent
  - motor\_settings\_t, [55](#)
- MaxCurrentConsumption
  - encoder\_settings\_t, [35](#)
  - hallsensor\_settings\_t, [48](#)
  - stage\_settings\_t, [69](#)
- MaxCurrentTime
  - motor\_settings\_t, [55](#)
- MaxOperatingFrequency
  - encoder\_settings\_t, [35](#)
  - hallsensor\_settings\_t, [48](#)
- MaxOutputBacklash
  - gear\_settings\_t, [44](#)
- MaxSpeed
  - control\_settings\_calb\_t, [25](#)
  - control\_settings\_t, [26](#)
  - motor\_settings\_t, [56](#)
  - stage\_settings\_t, [69](#)
- measurements\_t, [52](#)
  - Error, [53](#)
  - Length, [53](#)
  - Speed, [53](#)
- MechanicalTimeConstant
  - motor\_settings\_t, [56](#)
- MicrostepMode
  - engine\_settings\_calb\_t, [37](#)
  - engine\_settings\_t, [38](#)
- MinimumUusb
  - secure\_settings\_t, [64](#)
- Minor
  - device\_information\_t, [29](#)
  - serial\_number\_t, [65](#)
- motor\_information\_t, [53](#)
  - Manufacturer, [54](#)
  - PartNumber, [54](#)
- motor\_settings\_t, [54](#)
  - DetentTorque, [55](#)
  - MaxCurrent, [55](#)
  - MaxCurrentTime, [55](#)
  - MaxSpeed, [56](#)
  - MechanicalTimeConstant, [56](#)
  - MotorType, [56](#)
  - NoLoadCurrent, [56](#)
  - NoLoadSpeed, [56](#)
  - NominalCurrent, [56](#)
  - NominalPower, [56](#)
  - NominalSpeed, [56](#)
  - NominalTorque, [56](#)
  - NominalVoltage, [57](#)
  - Phases, [57](#)
  - Poles, [57](#)
  - RotorInertia, [57](#)
  - SpeedConstant, [57](#)
  - SpeedTorqueGradient, [57](#)
  - StallTorque, [57](#)
  - TorqueConstant, [57](#)
  - WindingInductance, [57](#)
  - WindingResistance, [58](#)
- MotorType
  - motor\_settings\_t, [56](#)
- move\_settings\_calb\_t, [58](#)
  - Accel, [58](#)
  - AntiplaySpeed, [58](#)
  - Decel, [58](#)
  - MoveFlags, [59](#)
  - Speed, [59](#)
- move\_settings\_t, [59](#)
  - Accel, [59](#)
  - AntiplaySpeed, [59](#)
  - Decel, [60](#)
  - MoveFlags, [60](#)
  - Speed, [60](#)
  - uAntiplaySpeed, [60](#)
  - uSpeed, [60](#)
- MoveFlags
  - move\_settings\_calb\_t, [59](#)
  - move\_settings\_t, [60](#)
- MoveSts
  - status\_calb\_t, [71](#)
  - status\_t, [74](#)
- msec\_sleep
  - ximc.h, [144](#)
- MvCmdSts
  - status\_calb\_t, [71](#)
  - status\_t, [74](#)
- NoLoadCurrent
  - motor\_settings\_t, [56](#)
- NoLoadSpeed
  - motor\_settings\_t, [56](#)
- NomCurrent
  - engine\_settings\_calb\_t, [37](#)
  - engine\_settings\_t, [39](#)
- NomSpeed
  - engine\_settings\_calb\_t, [37](#)
  - engine\_settings\_t, [39](#)
- NomVoltage
  - engine\_settings\_calb\_t, [37](#)

- engine\_settings\_t, [39](#)
- NominalCurrent
  - motor\_settings\_t, [56](#)
- NominalPower
  - motor\_settings\_t, [56](#)
- NominalSpeed
  - motor\_settings\_t, [56](#)
- NominalTorque
  - motor\_settings\_t, [56](#)
- NominalVoltage
  - motor\_settings\_t, [57](#)
- nonvolatile\_memory\_t, [60](#)
  - UserData, [61](#)
- open\_device
  - ximc.h, [144](#)
- POWER\_OFF\_ENABLED
  - ximc.h, [114](#)
- POWER\_REDUCT\_ENABLED
  - ximc.h, [114](#)
- POWER\_SMOOTH\_CURRENT
  - ximc.h, [114](#)
- PWR\_STATE\_MAX
  - ximc.h, [114](#)
- PWR\_STATE\_NORM
  - ximc.h, [114](#)
- PWR\_STATE\_OFF
  - ximc.h, [114](#)
- PWR\_STATE\_REDUCT
  - ximc.h, [114](#)
- PWR\_STATE\_UNKNOWN
  - ximc.h, [115](#)
- PWRSts
  - status\_calb\_t, [71](#)
  - status\_t, [74](#)
- PartNumber
  - encoder\_information\_t, [34](#)
  - gear\_information\_t, [43](#)
  - hallsensor\_information\_t, [47](#)
  - motor\_information\_t, [54](#)
  - stage\_information\_t, [67](#)
- Phases
  - motor\_settings\_t, [57](#)
- pid\_settings\_t, [61](#)
- Poles
  - motor\_settings\_t, [57](#)
- PosFlags
  - set\_position\_calb\_t, [65](#)
  - set\_position\_t, [66](#)
- Position
  - get\_position\_calb\_t, [45](#)
  - set\_position\_calb\_t, [65](#)
  - sync\_in\_settings\_calb\_t, [75](#)
- PositionerName
  - stage\_name\_t, [68](#)
- Pot
  - analog\_data\_t, [19](#)
  - chart\_data\_t, [23](#)
- power\_settings\_t, [61](#)
  - CurrReductDelay, [62](#)
  - CurrentSetTime, [62](#)
  - HoldCurrent, [62](#)
  - PowerFlags, [62](#)
  - PowerOffDelay, [62](#)
- PowerFlags
  - power\_settings\_t, [62](#)
- PowerOffDelay
  - power\_settings\_t, [62](#)
- probe\_device
  - ximc.h, [145](#)
- R
  - analog\_data\_t, [19](#)
  - emf\_settings\_t, [33](#)
- REV\_SENS\_INV
  - ximc.h, [115](#)
- RPM\_DIV\_1000
  - ximc.h, [115](#)
- RatedInputSpeed
  - gear\_settings\_t, [44](#)
- RatedInputTorque
  - gear\_settings\_t, [44](#)
- ReductionIn
  - gear\_settings\_t, [44](#)
- ReductionOut
  - gear\_settings\_t, [44](#)
- Release
  - device\_information\_t, [30](#)
  - serial\_number\_t, [65](#)
- reset\_locks
  - ximc.h, [145](#)
- RightBorder
  - edges\_settings\_calb\_t, [31](#)
  - edges\_settings\_t, [32](#)
- RotorInertia
  - motor\_settings\_t, [57](#)
- SN
  - serial\_number\_t, [65](#)
- STATE\_ALARM
  - ximc.h, [115](#)
- STATE\_BRAKE
  - ximc.h, [115](#)
- STATE\_BUTTON\_LEFT
  - ximc.h, [115](#)
- STATE\_BUTTON\_RIGHT
  - ximc.h, [115](#)
- STATE\_CONTR
  - ximc.h, [115](#)
- STATE\_CTP\_ERROR
  - ximc.h, [116](#)
- STATE\_DIG\_SIGNAL
  - ximc.h, [116](#)

STATE\_ENC\_A  
     ximc.h, [116](#)  
 STATE\_ENC\_B  
     ximc.h, [116](#)  
 STATE\_ERRC  
     ximc.h, [116](#)  
 STATE\_ERRD  
     ximc.h, [116](#)  
 STATE\_ERRV  
     ximc.h, [116](#)  
 STATE\_EXTIO\_ALARM  
     ximc.h, [116](#)  
 STATE\_GPIO\_LEVEL  
     ximc.h, [116](#)  
 STATE\_GPIO\_PINOUT  
     ximc.h, [117](#)  
 STATE\_LEFT\_EDGE  
     ximc.h, [117](#)  
 STATE\_POWER\_OVERHEAT  
     ximc.h, [117](#)  
 STATE\_REV\_SENSOR  
     ximc.h, [117](#)  
 STATE\_RIGHT\_EDGE  
     ximc.h, [117](#)  
 STATE\_SECUR  
     ximc.h, [117](#)  
 STATE\_SYNC\_INPUT  
     ximc.h, [117](#)  
 STATE\_SYNC\_OUTPUT  
     ximc.h, [118](#)  
 SYNCIN\_ENABLED  
     ximc.h, [118](#)  
 SYNCIN\_GOTOPOSITION  
     ximc.h, [118](#)  
 SYNCIN\_INVERT  
     ximc.h, [118](#)  
 SYNCOUT\_ENABLED  
     ximc.h, [118](#)  
 SYNCOUT\_IN\_STEPS  
     ximc.h, [118](#)  
 SYNCOUT\_INVERT  
     ximc.h, [118](#)  
 SYNCOUT\_ONPERIOD  
     ximc.h, [118](#)  
 SYNCOUT\_ONSTART  
     ximc.h, [118](#)  
 SYNCOUT\_ONSTOP  
     ximc.h, [118](#)  
 SYNCOUT\_STATE  
     ximc.h, [118](#)  
 secure\_settings\_t, [62](#)  
     Criticalpwr, [63](#)  
     Criticalusb, [63](#)  
     CriticalT, [63](#)  
     CriticalUpwr, [63](#)  
     CriticalUusb, [63](#)  
     Flags, [64](#)  
     LowUpwrOff, [64](#)  
     MinimumUusb, [64](#)  
 serial\_number\_t, [64](#)  
     Key, [64](#)  
     Major, [64](#)  
     Minor, [65](#)  
     Release, [65](#)  
     SN, [65](#)  
 service\_command\_updf  
     ximc.h, [145](#)  
 set\_accessories\_settings  
     ximc.h, [145](#)  
 set\_bindy\_key  
     ximc.h, [145](#)  
 set\_brake\_settings  
     ximc.h, [146](#)  
 set\_calibration\_settings  
     ximc.h, [146](#)  
 set\_control\_settings  
     ximc.h, [146](#)  
 set\_control\_settings\_calb  
     ximc.h, [146](#)  
 set\_controller\_name  
     ximc.h, [147](#)  
 set\_correction\_table  
     ximc.h, [147](#)  
 set\_ctp\_settings  
     ximc.h, [148](#)  
 set\_debug\_write  
     ximc.h, [148](#)  
 set\_edges\_settings  
     ximc.h, [148](#)  
 set\_edges\_settings\_calb  
     ximc.h, [148](#)  
 set\_emf\_settings  
     ximc.h, [149](#)  
 set\_encoder\_information  
     ximc.h, [149](#)  
 set\_encoder\_settings  
     ximc.h, [149](#)  
 set\_engine\_advansed\_setup  
     ximc.h, [149](#)  
 set\_engine\_settings  
     ximc.h, [150](#)  
 set\_engine\_settings\_calb  
     ximc.h, [150](#)  
 set\_entype\_settings  
     ximc.h, [150](#)  
 set\_extended\_settings  
     ximc.h, [151](#)  
 set\_extio\_settings  
     ximc.h, [151](#)  
 set\_feedback\_settings  
     ximc.h, [151](#)  
 set\_gear\_information

- ximc.h, [152](#)
- set\_gear\_settings
  - ximc.h, [152](#)
- set\_hallsensor\_information
  - ximc.h, [152](#)
- set\_hallsensor\_settings
  - ximc.h, [152](#)
- set\_home\_settings
  - ximc.h, [153](#)
- set\_home\_settings\_calb
  - ximc.h, [153](#)
- set\_joystick\_settings
  - ximc.h, [153](#)
- set\_logging\_callback
  - ximc.h, [154](#)
- set\_motor\_information
  - ximc.h, [154](#)
- set\_motor\_settings
  - ximc.h, [154](#)
- set\_move\_settings
  - ximc.h, [154](#)
- set\_move\_settings\_calb
  - ximc.h, [154](#)
- set\_nonvolatile\_memory
  - ximc.h, [155](#)
- set\_pid\_settings
  - ximc.h, [155](#)
- set\_position
  - ximc.h, [155](#)
- set\_position\_calb
  - ximc.h, [155](#)
- set\_position\_calb\_t, [65](#)
  - EncPosition, [65](#)
  - PosFlags, [65](#)
  - Position, [65](#)
- set\_position\_t, [66](#)
  - EncPosition, [66](#)
  - PosFlags, [66](#)
  - uPosition, [66](#)
- set\_power\_settings
  - ximc.h, [156](#)
- set\_secure\_settings
  - ximc.h, [156](#)
- set\_serial\_number
  - ximc.h, [156](#)
- set\_stage\_information
  - ximc.h, [156](#)
- set\_stage\_name
  - ximc.h, [157](#)
- set\_stage\_settings
  - ximc.h, [157](#)
- set\_sync\_in\_settings
  - ximc.h, [157](#)
- set\_sync\_in\_settings\_calb
  - ximc.h, [157](#)
- set\_sync\_out\_settings
  - ximc.h, [158](#)
- set\_sync\_out\_settings\_calb
  - ximc.h, [158](#)
- set\_uart\_settings
  - ximc.h, [159](#)
- SlowHome
  - home\_settings\_calb\_t, [49](#)
  - home\_settings\_t, [50](#)
- Speed
  - measurements\_t, [53](#)
  - move\_settings\_calb\_t, [59](#)
  - move\_settings\_t, [60](#)
  - sync\_in\_settings\_calb\_t, [75](#)
  - sync\_in\_settings\_t, [76](#)
- SpeedConstant
  - motor\_settings\_t, [57](#)
- SpeedTorqueGradient
  - motor\_settings\_t, [57](#)
- stage\_information\_t, [66](#)
  - Manufacturer, [67](#)
  - PartNumber, [67](#)
- stage\_name\_t, [67](#)
  - PositionerName, [68](#)
- stage\_settings\_t, [68](#)
  - HorizontalLoadCapacity, [68](#)
  - LeadScrewPitch, [68](#)
  - MaxCurrentConsumption, [69](#)
  - MaxSpeed, [69](#)
  - SupplyVoltageMax, [69](#)
  - SupplyVoltageMin, [69](#)
  - TravelRange, [69](#)
  - Units, [69](#)
  - VerticalLoadCapacity, [69](#)
- StallTorque
  - motor\_settings\_t, [57](#)
- status\_calb\_t, [69](#)
  - CmdBufFreeSpace, [70](#)
  - CurPosition, [70](#)
  - CurSpeed, [71](#)
  - CurT, [71](#)
  - EncPosition, [71](#)
  - EncSts, [71](#)
  - Flags, [71](#)
  - GPIOFlags, [71](#)
  - Ipwr, [71](#)
  - Iusb, [71](#)
  - MoveSts, [71](#)
  - MvCmdSts, [71](#)
  - PWRSts, [71](#)
  - Upwr, [71](#)
  - Uusb, [72](#)
  - WindSts, [72](#)
- status\_t, [72](#)
  - CmdBufFreeSpace, [73](#)
  - CurPosition, [73](#)
  - CurSpeed, [73](#)

- CurT, [73](#)
- EncPosition, [73](#)
- EncSts, [73](#)
- Flags, [73](#)
- GPIOFlags, [73](#)
- Ipwr, [74](#)
- Iusb, [74](#)
- MoveSts, [74](#)
- MvCmdSts, [74](#)
- PWRSts, [74](#)
- uCurPosition, [74](#)
- uCurSpeed, [74](#)
- Upwr, [74](#)
- Uusb, [74](#)
- WindSts, [74](#)
- stepcloseloop\_Kp\_high
  - engine.advanced\_setup\_t, [36](#)
- stepcloseloop\_Kp\_low
  - engine.advanced\_setup\_t, [36](#)
- stepcloseloop\_Kw
  - engine.advanced\_setup\_t, [36](#)
- StepsPerRev
  - engine.settings\_calb\_t, [37](#)
  - engine.settings\_t, [39](#)
- SupVoltage
  - analog\_data\_t, [19](#)
- SupVoltage\_ADC
  - analog\_data\_t, [19](#)
- SupplyVoltageMax
  - encoder.settings\_t, [35](#)
  - hallsensor.settings\_t, [48](#)
  - stage.settings\_t, [69](#)
- SupplyVoltageMin
  - encoder.settings\_t, [35](#)
  - hallsensor.settings\_t, [48](#)
  - stage.settings\_t, [69](#)
- sync\_in\_settings\_calb\_t, [75](#)
  - ClutterTime, [75](#)
  - Position, [75](#)
  - Speed, [75](#)
  - SyncInFlags, [75](#)
- sync\_in\_settings\_t, [75](#)
  - ClutterTime, [76](#)
  - Speed, [76](#)
  - SyncInFlags, [76](#)
  - uPosition, [76](#)
  - uSpeed, [76](#)
- sync\_out\_settings\_calb\_t, [77](#)
  - Accuracy, [77](#)
  - SyncOutFlags, [77](#)
  - SyncOutPeriod, [77](#)
  - SyncOutPulseSteps, [77](#)
- sync\_out\_settings\_t, [78](#)
  - Accuracy, [78](#)
  - SyncOutFlags, [78](#)
  - SyncOutPeriod, [78](#)
- SyncOutPulseSteps, [79](#)
  - uAccuracy, [79](#)
- SyncInFlags
  - sync\_in\_settings\_calb\_t, [75](#)
  - sync\_in\_settings\_t, [76](#)
- SyncOutFlags
  - sync\_out\_settings\_calb\_t, [77](#)
  - sync\_out\_settings\_t, [78](#)
- SyncOutPeriod
  - sync\_out\_settings\_calb\_t, [77](#)
  - sync\_out\_settings\_t, [78](#)
- SyncOutPulseSteps
  - sync\_out\_settings\_calb\_t, [77](#)
  - sync\_out\_settings\_t, [79](#)
- t1
  - brake.settings\_t, [20](#)
- t2
  - brake.settings\_t, [20](#)
- t3
  - brake.settings\_t, [20](#)
- t4
  - brake.settings\_t, [20](#)
- TSGrad
  - accessories.settings\_t, [15](#)
- TSMaX
  - accessories.settings\_t, [15](#)
- TSMIn
  - accessories.settings\_t, [15](#)
- TSSettings
  - accessories.settings\_t, [16](#)
- Temp
  - analog\_data\_t, [19](#)
- Temp\_ADC
  - analog\_data\_t, [19](#)
- TemperatureSensorInfo
  - accessories.settings\_t, [15](#)
- Timeout
  - control.settings\_calb\_t, [25](#)
  - control.settings\_t, [26](#)
- TorqueConstant
  - motor.settings\_t, [57](#)
- TravelRange
  - stage.settings\_t, [69](#)
- UART\_PARITY\_BITS
  - ximc.h, [119](#)
- UARTSetupFlags
  - uart.settings\_t, [79](#)
- uAccuracy
  - sync\_out\_settings\_t, [79](#)
- uAntiplaySpeed
  - move.settings\_t, [60](#)
- uCurPosition
  - status\_t, [74](#)
- uCurSpeed
  - status\_t, [74](#)

- uDeltaPosition
  - control\_settings\_t, [26](#)
- uFastHome
  - home\_settings\_t, [50](#)
- uHomeDelta
  - home\_settings\_t, [50](#)
- uLeftBorder
  - edges\_settings\_t, [32](#)
- uMaxSpeed
  - control\_settings\_t, [26](#)
- uNomSpeed
  - engine\_settings\_t, [39](#)
- uPosition
  - get\_position\_t, [45](#)
  - set\_position\_t, [66](#)
  - sync\_in\_settings\_t, [76](#)
- uRightBorder
  - edges\_settings\_t, [32](#)
- uSlowHome
  - home\_settings\_t, [50](#)
- uSpeed
  - move\_settings\_t, [60](#)
  - sync\_in\_settings\_t, [76](#)
- uart\_settings\_t, [79](#)
  - UARTSetupFlags, [79](#)
- UniquelD0
  - globally\_unique\_identifier\_t, [46](#)
- UniquelD1
  - globally\_unique\_identifier\_t, [46](#)
- UniquelD2
  - globally\_unique\_identifier\_t, [46](#)
- UniquelD3
  - globally\_unique\_identifier\_t, [46](#)
- Units
  - stage\_settings\_t, [69](#)
- Upwr
  - status\_calb\_t, [71](#)
  - status\_t, [74](#)
- UserData
  - nonvolatile\_memory\_t, [61](#)
- Uusb
  - status\_calb\_t, [72](#)
  - status\_t, [74](#)
- VerticalLoadCapacity
  - stage\_settings\_t, [69](#)
- WIND\_A\_STATE\_ABSENT
  - ximc.h, [119](#)
- WIND\_A\_STATE\_OK
  - ximc.h, [119](#)
- WIND\_B\_STATE\_ABSENT
  - ximc.h, [119](#)
- WIND\_B\_STATE\_OK
  - ximc.h, [119](#)
- WindSts
  - status\_calb\_t, [72](#)
  - status\_t, [74](#)
- WindingCurrentA
  - chart\_data\_t, [23](#)
- WindingCurrentB
  - chart\_data\_t, [23](#)
- WindingCurrentC
  - chart\_data\_t, [23](#)
- WindingInductance
  - motor\_settings\_t, [57](#)
- WindingResistance
  - motor\_settings\_t, [58](#)
- WindingVoltageA
  - chart\_data\_t, [23](#)
- WindingVoltageB
  - chart\_data\_t, [24](#)
- WindingVoltageC
  - chart\_data\_t, [24](#)
- write\_key
  - ximc.h, [159](#)
- XIMC\_API
  - ximc.h, [119](#)
- ximc.h, [80](#)
  - BACK\_EMF\_KM\_AUTO, [104](#)
  - BORDER\_IS\_ENCODER, [104](#)
  - BORDER\_STOP\_LEFT, [105](#)
  - BORDER\_STOP\_RIGHT, [105](#)
  - BRAKE\_ENABLED, [105](#)
  - BRAKE\_ENG\_PWROFF, [105](#)
  - CONTROL\_MODE\_BITS, [105](#)
  - CONTROL\_MODE\_JOY, [105](#)
  - CONTROL\_MODE\_LR, [105](#)
  - CONTROL\_MODE\_OFF, [105](#)
  - CTP\_ALARM\_ON\_ERROR, [105](#)
  - CTP\_BASE, [106](#)
  - CTP\_ENABLED, [106](#)
  - close\_device, [120](#)
  - command\_clear\_fram, [120](#)
  - command\_eeread\_settings, [120](#)
  - command\_eesave\_settings, [120](#)
  - command\_home, [121](#)
  - command\_homezero, [121](#)
  - command\_left, [121](#)
  - command\_loft, [121](#)
  - command\_move, [122](#)
  - command\_move\_calb, [122](#)
  - command\_movr, [122](#)
  - command\_movr\_calb, [123](#)
  - command\_power\_off, [123](#)
  - command\_read\_robust\_settings, [123](#)
  - command\_read\_settings, [123](#)
  - command\_reset, [124](#)
  - command\_right, [124](#)
  - command\_save\_robust\_settings, [124](#)
  - command\_save\_settings, [124](#)
  - command\_sstp, [124](#)

command\_start\_measurements, 125  
command\_stop, 125  
command\_update\_firmware, 125  
command\_wait\_for\_stop, 125  
command\_zero, 125  
EEPROM\_PRECEDENCE, 106  
ENC\_STATE\_ABSENT, 106  
ENC\_STATE\_MALFUNC, 106  
ENC\_STATE\_OK, 106  
ENC\_STATE\_REVERS, 106  
ENC\_STATE\_UNKNOWN, 107  
ENDER\_SWAP, 107  
ENGINE\_ACCEL\_ON, 107  
ENGINE\_ANTIPLAY, 107  
ENGINE\_LIMIT\_CURR, 107  
ENGINE\_LIMIT\_RPM, 107  
ENGINE\_LIMIT\_VOLT, 107  
ENGINE\_MAX\_SPEED, 108  
ENGINE\_REVERSE, 108  
ENGINE\_TYPE\_2DC, 108  
ENGINE\_TYPE\_DC, 108  
ENGINE\_TYPE\_NONE, 108  
ENGINE\_TYPE\_STEP, 108  
ENGINE\_TYPE\_TEST, 108  
ENUMERATE\_PROBE, 108  
EXTIO\_SETUP\_INVERT, 108  
EXTIO\_SETUP\_OUTPUT, 110  
enumerate\_devices, 126  
FEEDBACK\_EMF, 110  
FEEDBACK\_ENCODER, 110  
FEEDBACK\_NONE, 110  
free\_enumerate\_devices, 126  
get\_accessories\_settings, 126  
get\_analog\_data, 126  
get\_bootloader\_version, 127  
get\_brake\_settings, 127  
get\_calibration\_settings, 127  
get\_chart\_data, 127  
get\_control\_settings, 128  
get\_control\_settings\_calb, 128  
get\_controller\_name, 128  
get\_ctp\_settings, 129  
get\_debug\_read, 129  
get\_device\_count, 129  
get\_device\_information, 129  
get\_device\_name, 130  
get\_edges\_settings, 130  
get\_edges\_settings\_calb, 130  
get\_emf\_settings, 130  
get\_encoder\_information, 131  
get\_encoder\_settings, 131  
get\_engine\_advanced\_setup, 131  
get\_engine\_settings, 131  
get\_engine\_settings\_calb, 132  
get\_entype\_settings, 132  
get\_enumerate\_device\_controller\_name, 132  
get\_enumerate\_device\_information, 133  
get\_enumerate\_device\_network\_information, 133  
get\_enumerate\_device\_serial, 133  
get\_enumerate\_device\_stage\_name, 133  
get\_extended\_settings, 134  
get\_extio\_settings, 134  
get\_feedback\_settings, 134  
get\_firmware\_version, 135  
get\_gear\_information, 135  
get\_gear\_settings, 135  
get\_globally\_unique\_identifier, 135  
get\_hallsensor\_information, 136  
get\_hallsensor\_settings, 136  
get\_home\_settings, 136  
get\_home\_settings\_calb, 136  
get\_init\_random, 137  
get\_joystick\_settings, 137  
get\_measurements, 137  
get\_motor\_information, 137  
get\_motor\_settings, 138  
get\_move\_settings, 138  
get\_move\_settings\_calb, 138  
get\_nonvolatile\_memory, 138  
get\_pid\_settings, 138  
get\_position, 139  
get\_position\_calb, 139  
get\_power\_settings, 139  
get\_secure\_settings, 139  
get\_serial\_number, 140  
get\_stage\_information, 140  
get\_stage\_name, 140  
get\_stage\_settings, 140  
get\_status, 140  
get\_status\_calb, 141  
get\_sync\_in\_settings, 141  
get\_sync\_in\_settings\_calb, 141  
get\_sync\_out\_settings, 142  
get\_sync\_out\_settings\_calb, 142  
get\_uart\_settings, 142  
goto\_firmware, 143  
H\_BRIDGE\_ALERT, 110  
HOME\_DIR\_FIRST, 111  
HOME\_DIR\_SECOND, 111  
HOME\_HALF\_MV, 111  
HOME\_MV\_SEC\_EN, 111  
HOME\_USE\_FAST, 112  
has\_firmware, 143  
JOY\_REVERSE, 112  
LOW\_UPWR\_PROTECTION, 112  
load\_correction\_table, 143  
logging\_callback\_stderr\_narrow, 144  
logging\_callback\_stderr\_wide, 144  
logging\_callback\_t, 120  
MICROSTEP\_MODE\_FULL, 113  
MOVE\_STATE\_ANTIPLAY, 113  
MOVE\_STATE\_MOVING, 113

MVCMD\_ERROR, [113](#)  
MVCMD\_HOME, [113](#)  
MVCMD\_LEFT, [113](#)  
MVCMD\_LOFT, [113](#)  
MVCMD\_MOVE, [113](#)  
MVCMD\_MOVR, [113](#)  
MVCMD\_NAME\_BITS, [113](#)  
MVCMD\_RIGHT, [114](#)  
MVCMD\_RUNNING, [114](#)  
MVCMD\_SSTP, [114](#)  
MVCMD\_STOP, [114](#)  
MVCMD\_UKNWN, [114](#)  
msec\_sleep, [144](#)  
open\_device, [144](#)  
POWER\_OFF\_ENABLED, [114](#)  
PWR\_STATE\_MAX, [114](#)  
PWR\_STATE\_NORM, [114](#)  
PWR\_STATE\_OFF, [114](#)  
PWR\_STATE\_REDUCT, [114](#)  
PWR\_STATE\_UNKNOWN, [115](#)  
probe\_device, [145](#)  
REV\_SENS\_INV, [115](#)  
RPM\_DIV\_1000, [115](#)  
reset\_locks, [145](#)  
STATE\_ALARM, [115](#)  
STATE\_BRAKE, [115](#)  
STATE\_BUTTON\_LEFT, [115](#)  
STATE\_BUTTON\_RIGHT, [115](#)  
STATE\_CONTR, [115](#)  
STATE\_CTP\_ERROR, [116](#)  
STATE\_DIG\_SIGNAL, [116](#)  
STATE\_ENC\_A, [116](#)  
STATE\_ENC\_B, [116](#)  
STATE\_ERRC, [116](#)  
STATE\_ERRD, [116](#)  
STATE\_ERRV, [116](#)  
STATE\_EXTIO\_ALARM, [116](#)  
STATE\_GPIO\_LEVEL, [116](#)  
STATE\_GPIO\_PINOUT, [117](#)  
STATE\_LEFT\_EDGE, [117](#)  
STATE\_REV\_SENSOR, [117](#)  
STATE\_RIGHT\_EDGE, [117](#)  
STATE\_SECUR, [117](#)  
STATE\_SYNC\_INPUT, [117](#)  
STATE\_SYNC\_OUTPUT, [118](#)  
SYNCIN\_ENABLED, [118](#)  
SYNCIN\_GOTOPOSITION, [118](#)  
SYNCIN\_INVERT, [118](#)  
SYNCOUT\_ENABLED, [118](#)  
SYNCOUT\_IN\_STEPS, [118](#)  
SYNCOUT\_INVERT, [118](#)  
SYNCOUT\_ONPERIOD, [118](#)  
SYNCOUT\_ONSTART, [118](#)  
SYNCOUT\_ONSTOP, [118](#)  
SYNCOUT\_STATE, [118](#)  
service\_command\_updf, [145](#)  
set\_accessories\_settings, [145](#)  
set\_bindy\_key, [145](#)  
set\_brake\_settings, [146](#)  
set\_calibration\_settings, [146](#)  
set\_control\_settings, [146](#)  
set\_control\_settings\_calb, [146](#)  
set\_controller\_name, [147](#)  
set\_correction\_table, [147](#)  
set\_ctp\_settings, [148](#)  
set\_debug\_write, [148](#)  
set\_edges\_settings, [148](#)  
set\_edges\_settings\_calb, [148](#)  
set\_emf\_settings, [149](#)  
set\_encoder\_information, [149](#)  
set\_encoder\_settings, [149](#)  
set\_engine\_advanced\_setup, [149](#)  
set\_engine\_settings, [150](#)  
set\_engine\_settings\_calb, [150](#)  
set\_entype\_settings, [150](#)  
set\_extended\_settings, [151](#)  
set\_extio\_settings, [151](#)  
set\_feedback\_settings, [151](#)  
set\_gear\_information, [152](#)  
set\_gear\_settings, [152](#)  
set\_hallsensor\_information, [152](#)  
set\_hallsensor\_settings, [152](#)  
set\_home\_settings, [153](#)  
set\_home\_settings\_calb, [153](#)  
set\_joystick\_settings, [153](#)  
set\_logging\_callback, [154](#)  
set\_motor\_information, [154](#)  
set\_motor\_settings, [154](#)  
set\_move\_settings, [154](#)  
set\_move\_settings\_calb, [154](#)  
set\_nonvolatile\_memory, [155](#)  
set\_pid\_settings, [155](#)  
set\_position, [155](#)  
set\_position\_calb, [155](#)  
set\_power\_settings, [156](#)  
set\_secure\_settings, [156](#)  
set\_serial\_number, [156](#)  
set\_stage\_information, [156](#)  
set\_stage\_name, [157](#)  
set\_stage\_settings, [157](#)  
set\_sync\_in\_settings, [157](#)  
set\_sync\_in\_settings\_calb, [157](#)  
set\_sync\_out\_settings, [158](#)  
set\_sync\_out\_settings\_calb, [158](#)  
set\_uart\_settings, [159](#)  
UART\_PARITY\_BITS, [119](#)  
WIND\_A\_STATE\_OK, [119](#)  
WIND\_B\_STATE\_OK, [119](#)  
write\_key, [159](#)  
XIMC\_API, [119](#)  
ximc\_fix\_usbser\_sys, [159](#)  
ximc\_version, [159](#)

ximc\_fix\_usbser\_sys  
    ximc.h, [159](#)  
ximc\_version  
    ximc.h, [159](#)