

libximc
2.14.24

Generated by Doxygen 1.8.1.2

Wed Jul 17 2024 18:04:57

Contents

1	libximc library	1
1.1	What the controller does	1
1.2	What can do libximc library	1
1.3	Assistance	2
2	Introduction	3
2.1	About library	3
2.1.1	Supported OS and environment requirements:	3
3	How to rebuild library	4
3.1	Building on Windows	4
3.2	Building on debian-based linux systems	4
3.3	Building on MacOS X	4
3.4	Building on generic UNIX	5
3.5	Building on redhat-based linux systems	5
3.6	Source code access	5
4	How to use with...	6
4.1	Generic logging facility	9
4.2	Required permissions	9
4.3	C-profiles	9
4.4	Python-profiles	9
5	Working with user units	10
5.1	The structure of the conversion units calibration.t	10
5.2	Alternative functions for working with user units and data structures for them	10
5.3	Coordinate correction table for more accurate positioning	11
6	Data Structure Documentation	12
6.1	accessories_settings.t Struct Reference	12
6.1.1	Detailed Description	12
6.1.2	Field Documentation	13

6.1.2.1	LimitSwitchesSettings	13
6.1.2.2	MagneticBrakeInfo	13
6.1.2.3	MBRatedCurrent	13
6.1.2.4	MBRatedVoltage	13
6.1.2.5	MBSettings	13
6.1.2.6	MBTorque	13
6.1.2.7	TemperatureSensorInfo	13
6.1.2.8	TSGrad	13
6.1.2.9	TSMax	13
6.1.2.10	TSTMin	14
6.1.2.11	TSSettings	14
6.2	analog_data_t Struct Reference	14
6.2.1	Detailed Description	15
6.2.2	Field Documentation	15
6.2.2.1	A1Voltage	15
6.2.2.2	A1Voltage_ADC	15
6.2.2.3	A2Voltage	15
6.2.2.4	A2Voltage_ADC	16
6.2.2.5	ACurrent	16
6.2.2.6	ACurrent_ADC	16
6.2.2.7	B1Voltage	16
6.2.2.8	B1Voltage_ADC	16
6.2.2.9	B2Voltage	16
6.2.2.10	B2Voltage_ADC	16
6.2.2.11	BCurrent	16
6.2.2.12	BCurrent_ADC	16
6.2.2.13	FullCurrent	16
6.2.2.14	FullCurrent_ADC	16
6.2.2.15	H5	16
6.2.2.16	Joy	17
6.2.2.17	Joy_ADC	17
6.2.2.18	L	17
6.2.2.19	L5	17
6.2.2.20	L5_ADC	17
6.2.2.21	Pot	17
6.2.2.22	R	17
6.2.2.23	SupVoltage	17
6.2.2.24	SupVoltage_ADC	17
6.2.2.25	Temp	17

6.2.2.26	Temp_ADC	17
6.3	brake_settings_t Struct Reference	18
6.3.1	Detailed Description	18
6.3.2	Field Documentation	18
6.3.2.1	BrakeFlags	18
6.3.2.2	t1	18
6.3.2.3	t2	18
6.3.2.4	t3	18
6.3.2.5	t4	18
6.4	calibration_settings_t Struct Reference	19
6.4.1	Detailed Description	19
6.4.2	Field Documentation	19
6.4.2.1	CSS1_A	19
6.4.2.2	CSS1_B	19
6.4.2.3	CSS2_A	19
6.4.2.4	CSS2_B	19
6.4.2.5	FullCurrent_A	20
6.4.2.6	FullCurrent_B	20
6.5	calibration_t Struct Reference	20
6.5.1	Detailed Description	20
6.6	chart_data_t Struct Reference	20
6.6.1	Detailed Description	21
6.6.2	Field Documentation	21
6.6.2.1	AveragedPowerRatio	21
6.6.2.2	Joy	21
6.6.2.3	Pot	21
6.6.2.4	WindingCurrentA	21
6.6.2.5	WindingCurrentB	21
6.6.2.6	WindingCurrentC	21
6.6.2.7	WindingVoltageA	22
6.6.2.8	WindingVoltageB	22
6.6.2.9	WindingVoltageC	22
6.7	control_settings_calb_t Struct Reference	22
6.7.1	Detailed Description	22
6.7.2	Field Documentation	23
6.7.2.1	Flags	23
6.7.2.2	MaxClickTime	23
6.7.2.3	MaxSpeed	23
6.7.2.4	Timeout	23

6.8	control_settings_t Struct Reference	23
6.8.1	Detailed Description	23
6.8.2	Field Documentation	24
6.8.2.1	Flags	24
6.8.2.2	MaxClickTime	24
6.8.2.3	MaxSpeed	24
6.8.2.4	Timeout	24
6.8.2.5	uDeltaPosition	24
6.8.2.6	uMaxSpeed	24
6.9	controller_name_t Struct Reference	25
6.9.1	Detailed Description	25
6.9.2	Field Documentation	25
6.9.2.1	ControllerName	25
6.9.2.2	CtrlFlags	25
6.10	ctp_settings_t Struct Reference	25
6.10.1	Detailed Description	25
6.10.2	Field Documentation	26
6.10.2.1	CTPFlags	26
6.10.2.2	CTPMinError	26
6.11	debug_read_t Struct Reference	26
6.11.1	Detailed Description	26
6.11.2	Field Documentation	26
6.11.2.1	DebugData	26
6.12	debug_write_t Struct Reference	27
6.12.1	Detailed Description	27
6.12.2	Field Documentation	27
6.12.2.1	DebugData	27
6.13	device_information_t Struct Reference	27
6.13.1	Detailed Description	27
6.13.2	Field Documentation	28
6.13.2.1	Major	28
6.13.2.2	Minor	28
6.13.2.3	Release	28
6.14	device_network_information_t Struct Reference	28
6.14.1	Detailed Description	28
6.15	edges_settings_calb_t Struct Reference	28
6.15.1	Detailed Description	29
6.15.2	Field Documentation	29
6.15.2.1	BorderFlags	29

6.15.2.2	EnderFlags	29
6.15.2.3	LeftBorder	29
6.15.2.4	RightBorder	29
6.16	edges_settings_t Struct Reference	29
6.16.1	Detailed Description	30
6.16.2	Field Documentation	30
6.16.2.1	BorderFlags	30
6.16.2.2	EnderFlags	30
6.16.2.3	LeftBorder	30
6.16.2.4	RightBorder	30
6.16.2.5	uLeftBorder	30
6.16.2.6	uRightBorder	31
6.17	emf_settings_t Struct Reference	31
6.17.1	Detailed Description	31
6.17.2	Field Documentation	31
6.17.2.1	BackEMFFlags	31
6.17.2.2	Km	31
6.17.2.3	L	31
6.17.2.4	R	32
6.18	encoder_information_t Struct Reference	32
6.18.1	Detailed Description	32
6.18.2	Field Documentation	32
6.18.2.1	Manufacturer	32
6.18.2.2	PartNumber	32
6.19	encoder_settings_t Struct Reference	32
6.19.1	Detailed Description	33
6.19.2	Field Documentation	33
6.19.2.1	EncoderSettings	33
6.19.2.2	MaxCurrentConsumption	33
6.19.2.3	MaxOperatingFrequency	33
6.19.2.4	SupplyVoltageMax	33
6.19.2.5	SupplyVoltageMin	33
6.20	engine_advansed_setup_t Struct Reference	34
6.20.1	Detailed Description	34
6.20.2	Field Documentation	34
6.20.2.1	stepcloseloop_Kp_high	34
6.20.2.2	stepcloseloop_Kp_low	34
6.20.2.3	stepcloseloop_Kw	34
6.21	engine_settings_calb_t Struct Reference	34

6.21.1 Detailed Description	35
6.21.2 Field Documentation	35
6.21.2.1 Antiplay	35
6.21.2.2 EngineFlags	35
6.21.2.3 MicrostepMode	35
6.21.2.4 NomCurrent	35
6.21.2.5 NomSpeed	36
6.21.2.6 NomVoltage	36
6.21.2.7 StepsPerRev	36
6.22 engine_settings_t Struct Reference	36
6.22.1 Detailed Description	36
6.22.2 Field Documentation	37
6.22.2.1 Antiplay	37
6.22.2.2 EngineFlags	37
6.22.2.3 MicrostepMode	37
6.22.2.4 NomCurrent	37
6.22.2.5 NomSpeed	37
6.22.2.6 NomVoltage	37
6.22.2.7 StepsPerRev	37
6.22.2.8 uNomSpeed	37
6.23 entype_settings_t Struct Reference	38
6.23.1 Detailed Description	38
6.23.2 Field Documentation	38
6.23.2.1 DriverType	38
6.23.2.2 EngineType	38
6.24 extended_settings_t Struct Reference	38
6.24.1 Detailed Description	38
6.25 extio_settings_t Struct Reference	39
6.25.1 Detailed Description	39
6.25.2 Field Documentation	39
6.25.2.1 EXTIOModeFlags	39
6.25.2.2 EXTIOSetupFlags	39
6.26 feedback_settings_t Struct Reference	39
6.26.1 Detailed Description	40
6.26.2 Field Documentation	40
6.26.2.1 CountsPerTurn	40
6.26.2.2 FeedbackFlags	40
6.26.2.3 FeedbackType	40
6.26.2.4 IPS	40

6.27 gear_information_t Struct Reference	40
6.27.1 Detailed Description	41
6.27.2 Field Documentation	41
6.27.2.1 Manufacturer	41
6.27.2.2 PartNumber	41
6.28 gear_settings_t Struct Reference	41
6.28.1 Detailed Description	41
6.28.2 Field Documentation	42
6.28.2.1 Efficiency	42
6.28.2.2 InputInertia	42
6.28.2.3 MaxOutputBacklash	42
6.28.2.4 RatedInputSpeed	42
6.28.2.5 RatedInputTorque	42
6.28.2.6 ReductionIn	42
6.28.2.7 ReductionOut	42
6.29 get_position_calb_t Struct Reference	42
6.29.1 Detailed Description	43
6.29.2 Field Documentation	43
6.29.2.1 EncPosition	43
6.29.2.2 Position	43
6.30 get_position_t Struct Reference	43
6.30.1 Detailed Description	43
6.30.2 Field Documentation	44
6.30.2.1 EncPosition	44
6.30.2.2 uPosition	44
6.31 globally_unique_identifier_t Struct Reference	44
6.31.1 Detailed Description	44
6.31.2 Field Documentation	44
6.31.2.1 UniqueID0	44
6.31.2.2 UniqueID1	44
6.31.2.3 UniqueID2	44
6.31.2.4 UniqueID3	45
6.32 hallsensor_information_t Struct Reference	45
6.32.1 Detailed Description	45
6.32.2 Field Documentation	45
6.32.2.1 Manufacturer	45
6.32.2.2 PartNumber	45
6.33 hallsensor_settings_t Struct Reference	45
6.33.1 Detailed Description	46

6.33.2 Field Documentation	46
6.33.2.1 MaxCurrentConsumption	46
6.33.2.2 MaxOperatingFrequency	46
6.33.2.3 SupplyVoltageMax	46
6.33.2.4 SupplyVoltageMin	46
6.34 home_settings_calb_t Struct Reference	46
6.34.1 Detailed Description	47
6.34.2 Field Documentation	47
6.34.2.1 FastHome	47
6.34.2.2 HomeDelta	47
6.34.2.3 HomeFlags	47
6.34.2.4 SlowHome	47
6.35 home_settings_t Struct Reference	47
6.35.1 Detailed Description	48
6.35.2 Field Documentation	48
6.35.2.1 FastHome	48
6.35.2.2 HomeDelta	48
6.35.2.3 HomeFlags	48
6.35.2.4 SlowHome	48
6.35.2.5 uFastHome	49
6.35.2.6 uHomeDelta	49
6.35.2.7 uSlowHome	49
6.36 init_random_t Struct Reference	49
6.36.1 Detailed Description	49
6.36.2 Field Documentation	49
6.36.2.1 key	49
6.37 joystick_settings_t Struct Reference	49
6.37.1 Detailed Description	50
6.37.2 Field Documentation	50
6.37.2.1 DeadZone	50
6.37.2.2 ExpFactor	50
6.37.2.3 JoyCenter	50
6.37.2.4 JoyFlags	51
6.37.2.5 JoyHighEnd	51
6.37.2.6 JoyLowEnd	51
6.38 measurements_t Struct Reference	51
6.38.1 Detailed Description	51
6.38.2 Field Documentation	51
6.38.2.1 Error	51

6.38.2.2 Length	51
6.38.2.3 Speed	52
6.39 motor_information_t Struct Reference	52
6.39.1 Detailed Description	52
6.39.2 Field Documentation	52
6.39.2.1 Manufacturer	52
6.39.2.2 PartNumber	52
6.40 motor_settings_t Struct Reference	52
6.40.1 Detailed Description	54
6.40.2 Field Documentation	54
6.40.2.1 DetentTorque	54
6.40.2.2 MaxCurrent	54
6.40.2.3 MaxCurrentTime	54
6.40.2.4 MaxSpeed	54
6.40.2.5 MechanicalTimeConstant	54
6.40.2.6 MotorType	54
6.40.2.7 NoLoadCurrent	54
6.40.2.8 NoLoadSpeed	55
6.40.2.9 NominalCurrent	55
6.40.2.10NominalPower	55
6.40.2.11NominalSpeed	55
6.40.2.12NominalTorque	55
6.40.2.13NominalVoltage	55
6.40.2.14Phases	55
6.40.2.15Poles	55
6.40.2.16RotorInertia	55
6.40.2.17SpeedConstant	56
6.40.2.18SpeedTorqueGradient	56
6.40.2.19StallTorque	56
6.40.2.20TorqueConstant	56
6.40.2.21WindingInductance	56
6.40.2.22WindingResistance	56
6.41 move_settings_calb_t Struct Reference	56
6.41.1 Detailed Description	57
6.41.2 Field Documentation	57
6.41.2.1 Accel	57
6.41.2.2 AntiplaySpeed	57
6.41.2.3 Decel	57
6.41.2.4 MoveFlags	57

6.41.2.5 Speed	57
6.42 move_settings_t Struct Reference	57
6.42.1 Detailed Description	58
6.42.2 Field Documentation	58
6.42.2.1 Accel	58
6.42.2.2 AntiplaySpeed	58
6.42.2.3 Decel	58
6.42.2.4 MoveFlags	58
6.42.2.5 Speed	58
6.42.2.6 uAntiplaySpeed	59
6.42.2.7 uSpeed	59
6.43 network_settings_t Struct Reference	59
6.43.1 Detailed Description	59
6.43.2 Field Documentation	59
6.43.2.1 DefaultGateway	59
6.43.2.2 DHCPEnabled	59
6.43.2.3 IPv4Address	60
6.43.2.4 SubnetMask	60
6.44 nonvolatile_memory_t Struct Reference	60
6.44.1 Detailed Description	60
6.44.2 Field Documentation	60
6.44.2.1 UserData	60
6.45 password_settings_t Struct Reference	60
6.45.1 Detailed Description	60
6.45.2 Field Documentation	61
6.45.2.1 UserPassword	61
6.46 pid_settings_t Struct Reference	61
6.46.1 Detailed Description	61
6.47 power_settings_t Struct Reference	61
6.47.1 Detailed Description	62
6.47.2 Field Documentation	62
6.47.2.1 CurrentSetTime	62
6.47.2.2 CurrReductDelay	62
6.47.2.3 HoldCurrent	62
6.47.2.4 PowerFlags	62
6.47.2.5 PowerOffDelay	62
6.48 secure_settings_t Struct Reference	62
6.48.1 Detailed Description	63
6.48.2 Field Documentation	63

6.48.2.1	CriticalUpwr	63
6.48.2.2	CriticalUsb	63
6.48.2.3	CriticalT	63
6.48.2.4	CriticalUpwr	63
6.48.2.5	CriticalUsb	64
6.48.2.6	Flags	64
6.48.2.7	LowUpwrOff	64
6.48.2.8	MinimumUsb	64
6.49	serial_number_t Struct Reference	64
6.49.1	Detailed Description	64
6.49.2	Field Documentation	64
6.49.2.1	Key	64
6.49.2.2	Major	65
6.49.2.3	Minor	65
6.49.2.4	Release	65
6.49.2.5	SN	65
6.50	set_position_calb_t Struct Reference	65
6.50.1	Detailed Description	65
6.50.2	Field Documentation	65
6.50.2.1	EncPosition	65
6.50.2.2	PosFlags	65
6.50.2.3	Position	66
6.51	set_position_t Struct Reference	66
6.51.1	Detailed Description	66
6.51.2	Field Documentation	66
6.51.2.1	EncPosition	66
6.51.2.2	PosFlags	66
6.51.2.3	uPosition	66
6.52	stage_information_t Struct Reference	66
6.52.1	Detailed Description	67
6.52.2	Field Documentation	67
6.52.2.1	Manufacturer	67
6.52.2.2	PartNumber	67
6.53	stage_name_t Struct Reference	67
6.53.1	Detailed Description	67
6.53.2	Field Documentation	68
6.53.2.1	PositionerName	68
6.54	stage_settings_t Struct Reference	68
6.54.1	Detailed Description	68

6.54.2	Field Documentation	68
6.54.2.1	HorizontalLoadCapacity	68
6.54.2.2	LeadScrewPitch	69
6.54.2.3	MaxCurrentConsumption	69
6.54.2.4	MaxSpeed	69
6.54.2.5	SupplyVoltageMax	69
6.54.2.6	SupplyVoltageMin	69
6.54.2.7	TravelRange	69
6.54.2.8	Units	69
6.54.2.9	VerticalLoadCapacity	69
6.55	status_calb_t Struct Reference	69
6.55.1	Detailed Description	70
6.55.2	Field Documentation	70
6.55.2.1	CmdBufFreeSpace	70
6.55.2.2	CurPosition	71
6.55.2.3	CurSpeed	71
6.55.2.4	CurT	71
6.55.2.5	EncPosition	71
6.55.2.6	EncSts	71
6.55.2.7	Flags	71
6.55.2.8	GPIOFlags	71
6.55.2.9	Ipwr	71
6.55.2.10	Iusb	71
6.55.2.11	MoveSts	71
6.55.2.12	MvCmdSts	71
6.55.2.13	PWRSts	71
6.55.2.14	Upwr	72
6.55.2.15	Uusb	72
6.55.2.16	WindSts	72
6.56	status_t Struct Reference	72
6.56.1	Detailed Description	73
6.56.2	Field Documentation	73
6.56.2.1	CmdBufFreeSpace	73
6.56.2.2	CurPosition	73
6.56.2.3	CurSpeed	73
6.56.2.4	CurT	73
6.56.2.5	EncPosition	73
6.56.2.6	EncSts	73
6.56.2.7	Flags	73

6.56.2.8	GPIOFlags	74
6.56.2.9	Ipwr	74
6.56.2.10	Iusb	74
6.56.2.11	MoveSts	74
6.56.2.12	MvCmdSts	74
6.56.2.13	PWRSts	74
6.56.2.14	uCurPosition	74
6.56.2.15	uCurSpeed	74
6.56.2.16	Upwr	74
6.56.2.17	Uusb	74
6.56.2.18	WindSts	74
6.57	sync_in_settings_calb_t Struct Reference	75
6.57.1	Detailed Description	75
6.57.2	Field Documentation	75
6.57.2.1	ClutterTime	75
6.57.2.2	Position	75
6.57.2.3	Speed	75
6.57.2.4	SyncInFlags	75
6.58	sync_in_settings_t Struct Reference	75
6.58.1	Detailed Description	76
6.58.2	Field Documentation	76
6.58.2.1	ClutterTime	76
6.58.2.2	Speed	76
6.58.2.3	SyncInFlags	76
6.58.2.4	uPosition	76
6.58.2.5	uSpeed	77
6.59	sync_out_settings_calb_t Struct Reference	77
6.59.1	Detailed Description	77
6.59.2	Field Documentation	77
6.59.2.1	Accuracy	77
6.59.2.2	SyncOutFlags	77
6.59.2.3	SyncOutPeriod	77
6.59.2.4	SyncOutPulseSteps	78
6.60	sync_out_settings_t Struct Reference	78
6.60.1	Detailed Description	78
6.60.2	Field Documentation	78
6.60.2.1	Accuracy	78
6.60.2.2	SyncOutFlags	78
6.60.2.3	SyncOutPeriod	79

6.60.2.4 SyncOutPulseSteps	79
6.60.2.5 uAccuracy	79
6.61 uart_settings_t Struct Reference	79
6.61.1 Detailed Description	79
6.61.2 Field Documentation	79
6.61.2.1 UARTSetupFlags	79
7 File Documentation	80
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	105
7.1.2.3 BACK_EMF_KM_AUTO	105
7.1.2.4 BACK_EMF_RESISTANCE_AUTO	105
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	106
7.1.2.15 CONTROL_MODE_LR	106
7.1.2.16 CONTROL_MODE_OFF	106
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	107
7.1.2.26 ENC_STATE_MALFUNC	107
7.1.2.27 ENC_STATE_OK	107
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	108
7.1.2.37	ENGINE_LIMIT_RPM	108
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	109
7.1.2.47	ENUMERATE_PROBE	109
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	110
7.1.2.58	EXTIO_SETUP_MODE_OUT_MOTOR_ON	110
7.1.2.59	EXTIO_SETUP_MODE_OUT_MOVING	110
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	111
7.1.2.70	FEEDBACK_ENCODER_MEDIATED	111
7.1.2.71	FEEDBACK_NONE	111
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	112
7.1.2.81	HOME_STOP_SECOND_BITS	112
7.1.2.82	HOME_STOP_SECOND_LIM	112
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	113
7.1.2.93	MICROSTEP_MODE_FRAC_4	113
7.1.2.94	MICROSTEP_MODE_FRAC_64	113
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	114
7.1.2.104	MVCMD_MOVE	114
7.1.2.105	MVCMD_MOVR	114
7.1.2.106	MVCMD_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	115
7.1.2.116PWR_STATE_NORM	115
7.1.2.117PWR_STATE_OFF	115
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	116
7.1.2.127STATE_BUTTON_LEFT	116
7.1.2.128STATE_BUTTON_RIGHT	116
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	117
7.1.2.137STATE_ERRC	117
7.1.2.138STATE_ERRD	117
7.1.2.139STATE_ERRV	117
7.1.2.140STATE_EXTIO_ALARM	117
7.1.2.141STATE_GPIO_LEVEL	117
7.1.2.142STATE_GPIO_PINOUT	117
7.1.2.143STATE_IS_HOMED	117
7.1.2.144STATE_LEFT_EDGE	117
7.1.2.145STATE_LOW_USB_VOLTAGE	118

7.1.2.146	STATE_OVERLOAD_POWER_CURRENT	118
7.1.2.147	STATE_OVERLOAD_POWER_VOLTAGE	118
7.1.2.148	STATE_OVERLOAD_USB_CURRENT	118
7.1.2.149	STATE_OVERLOAD_USB_VOLTAGE	118
7.1.2.150	STATE_POWER_OVERHEAT	118
7.1.2.151	STATE_REV_SENSOR	118
7.1.2.152	STATE_RIGHT_EDGE	118
7.1.2.153	STATE_SECUR	118
7.1.2.154	STATE_SYNC_INPUT	118
7.1.2.155	STATE_SYNC_OUTPUT	118
7.1.2.156	STATE_WINDING_RES_MISMATCH	119
7.1.2.157	SYNCIN_ENABLED	119
7.1.2.158	SYNCIN_GOTOPOSITION	119
7.1.2.159	SYNCIN_INVERT	119
7.1.2.160	SYNCOUT_ENABLED	119
7.1.2.161	SYNCOUT_IN_STEPS	119
7.1.2.162	SYNCOUT_INVERT	119
7.1.2.163	SYNCOUT_ONPERIOD	119
7.1.2.164	SYNCOUT_ONSTART	119
7.1.2.165	SYNCOUT_ONSTOP	119
7.1.2.166	SYNCOUT_STATE	119
7.1.2.167	UART_PARITY_BITS	120
7.1.2.168	WIND_A.STATE_ABSENT	120
7.1.2.169	WIND_A.STATE_MALFUNC	120
7.1.2.170	WIND_A.STATE_OK	120
7.1.2.171	WIND_A.STATE_UNKNOWN	120
7.1.2.172	WIND_B.STATE_ABSENT	120
7.1.2.173	WIND_B.STATE_MALFUNC	120
7.1.2.174	WIND_B.STATE_OK	120
7.1.2.175	WIND_B.STATE_UNKNOWN	120
7.1.2.176	XIMC_API	120
7.1.3	Typedef Documentation	120
7.1.3.1	logging_callback_t	120
7.1.4	Function Documentation	121
7.1.4.1	close_device	121
7.1.4.2	command_clear_fram	121
7.1.4.3	command_eeread_settings	121
7.1.4.4	command_eesave_settings	121
7.1.4.5	command_home	121

7.1.4.6	command_homezero	122
7.1.4.7	command_left	122
7.1.4.8	command_loft	122
7.1.4.9	command_move	123
7.1.4.10	command_move_calb	123
7.1.4.11	command_movr	123
7.1.4.12	command_movr_calb	124
7.1.4.13	command_power_off	124
7.1.4.14	command_read_robust_settings	124
7.1.4.15	command_read_settings	124
7.1.4.16	command_reset	125
7.1.4.17	command_right	125
7.1.4.18	command_save_robust_settings	125
7.1.4.19	command_save_settings	125
7.1.4.20	command_sstp	125
7.1.4.21	command_start_measurements	126
7.1.4.22	command_stop	126
7.1.4.23	command_update_firmware	126
7.1.4.24	command_wait_for_stop	126
7.1.4.25	command_zero	127
7.1.4.26	enumerate_devices	127
7.1.4.27	free_enumerate_devices	127
7.1.4.28	get_accessories_settings	127
7.1.4.29	get_analog_data	128
7.1.4.30	get_bootloader_version	128
7.1.4.31	get_brake_settings	128
7.1.4.32	get_calibration_settings	128
7.1.4.33	get_chart_data	129
7.1.4.34	get_control_settings	129
7.1.4.35	get_control_settings_calb	129
7.1.4.36	get_controller_name	130
7.1.4.37	get_ctp_settings	130
7.1.4.38	get_debug_read	130
7.1.4.39	get_device_count	130
7.1.4.40	get_device_information	130
7.1.4.41	get_device_name	131
7.1.4.42	get_edges_settings	131
7.1.4.43	get_edges_settings_calb	131
7.1.4.44	get_emf_settings	132

7.1.4.45	get_encoder_information	132
7.1.4.46	get_encoder_settings	132
7.1.4.47	get_engine_advanced_setup	132
7.1.4.48	get_engine_settings	133
7.1.4.49	get_engine_settings_calb	133
7.1.4.50	get_entype_settings	133
7.1.4.51	get_enumerate_device_controller_name	134
7.1.4.52	get_enumerate_device_information	134
7.1.4.53	get_enumerate_device_network_information	134
7.1.4.54	get_enumerate_device_serial	134
7.1.4.55	get_enumerate_device_stage_name	135
7.1.4.56	get_extended_settings	135
7.1.4.57	get_extio_settings	135
7.1.4.58	get_feedback_settings	136
7.1.4.59	get_firmware_version	136
7.1.4.60	get_gear_information	136
7.1.4.61	get_gear_settings	136
7.1.4.62	get_globally_unique_identifier	137
7.1.4.63	get_hallsensor_information	137
7.1.4.64	get_hallsensor_settings	137
7.1.4.65	get_home_settings	137
7.1.4.66	get_home_settings_calb	138
7.1.4.67	get_init_random	138
7.1.4.68	get_joystick_settings	138
7.1.4.69	get_measurements	138
7.1.4.70	get_motor_information	139
7.1.4.71	get_motor_settings	139
7.1.4.72	get_move_settings	139
7.1.4.73	get_move_settings_calb	139
7.1.4.74	get_network_settings	140
7.1.4.75	get_nonvolatile_memory	140
7.1.4.76	get_password_settings	140
7.1.4.77	get_pid_settings	140
7.1.4.78	get_position	141
7.1.4.79	get_position_calb	141
7.1.4.80	get_power_settings	141
7.1.4.81	get_secure_settings	142
7.1.4.82	get_serial_number	142
7.1.4.83	get_stage_information	142

7.1.4.84	get_stage_name	142
7.1.4.85	get_stage_settings	142
7.1.4.86	get_status	143
7.1.4.87	get_status_calb	143
7.1.4.88	get_sync_in_settings	143
7.1.4.89	get_sync_in_settings_calb	144
7.1.4.90	get_sync_out_settings	144
7.1.4.91	get_sync_out_settings_calb	144
7.1.4.92	get_uart_settings	145
7.1.4.93	goto_firmware	145
7.1.4.94	has_firmware	145
7.1.4.95	load_correction_table	145
7.1.4.96	logging_callback_stderr_narrow	146
7.1.4.97	logging_callback_stderr_wide	146
7.1.4.98	msec_sleep	146
7.1.4.99	open_device	146
7.1.4.100	probe_device	147
7.1.4.101	service_command_updf	147
7.1.4.102	set_accessories_settings	147
7.1.4.103	set_bindy_key	147
7.1.4.104	set_brake_settings	148
7.1.4.105	set_calibration_settings	148
7.1.4.106	set_control_settings	148
7.1.4.107	set_control_settings_calb	148
7.1.4.108	set_controller_name	149
7.1.4.109	set_correction_table	149
7.1.4.110	set_ctp_settings	150
7.1.4.111	set_debug_write	150
7.1.4.112	set_edges_settings	150
7.1.4.113	set_edges_settings_calb	150
7.1.4.114	set_emf_settings	151
7.1.4.115	set_encoder_information	151
7.1.4.116	set_encoder_settings	151
7.1.4.117	set_engine_advanced_setup	152
7.1.4.118	set_engine_settings	152
7.1.4.119	set_engine_settings_calb	152
7.1.4.120	set_entype_settings	153
7.1.4.121	set_extended_settings	153
7.1.4.122	set_extio_settings	153

7.1.4.123set_feedback_settings	154
7.1.4.124set_gear_information	154
7.1.4.125set_gear_settings	154
7.1.4.126set_hallsensor_information	154
7.1.4.127set_hallsensor_settings	155
7.1.4.128set_home_settings	155
7.1.4.129set_home_settings_calb	155
7.1.4.130set_joystick_settings	155
7.1.4.131set_logging_callback	156
7.1.4.132set_motor_information	156
7.1.4.133set_motor_settings	156
7.1.4.134set_move_settings	156
7.1.4.135set_move_settings_calb	157
7.1.4.136set_network_settings	157
7.1.4.137set_nonvolatile_memory	157
7.1.4.138set_password_settings	157
7.1.4.139set_pid_settings	158
7.1.4.140set_position	158
7.1.4.141set_position_calb	158
7.1.4.142set_power_settings	159
7.1.4.143set_secure_settings	159
7.1.4.144set_serial_number	159
7.1.4.145set_stage_information	159
7.1.4.146set_stage_name	160
7.1.4.147set_stage_settings	160
7.1.4.148set_sync_in_settings	160
7.1.4.149set_sync_in_settings_calb	160
7.1.4.150set_sync_out_settings	161
7.1.4.151set_sync_out_settings_calb	161
7.1.4.152set_uart_settings	161
7.1.4.153write_key	162
7.1.4.154ximc_fix_usbser_sys	162
7.1.4.155ximc_version	162

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 controller using ready-made [xilab software](#) or using examples which allow rapid development using C++, C#, .NET, Delphi, Visual Basic, Xcode, Python, Matlab, Java, LabWindows and LabVIEW.

1.2 What can do libximc library

- Libximc manages controller using interfaces: USB 2.0, RS232 and Ethernet, also uses a common and proven virtual serial port interface, so you can work with motor control modules through this library under almost all operating systems, including Windows, Linux and MacOS 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 us **errors** and **suggestions**. We appreciate your suggestions and try to make our product better!

Chapter 2

Introduction

2.1 About library

This document contains all information about libximc library. It utilizes well known virtual COM-port interface, so you can use it on Windows, Linux, MacOS X for Intel and Apple Silicon (via Rosetta 2) including 64-bit versions. Multi-platform programming library supports plug/unplug on the fly.

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

2.1.1 Supported OS and environment requirements:

- MacOS X 10.6 or newer
- Windows 2000 or newer
- Linux debian-based. 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

Build requirements:

- Windows: Microsoft Visual C++ 2013 or newer, MATLAB, Code::Blocks, Delphi, Java, Python, cygwin with tar, bison, flex, curl, 7z mingw
- UNIX: gcc 4 or newer, gmake, doxygen, LaTeX, flex 2.5.30+, bison 2.3+, autotools (autoconf, autoheader, aclocal, automake, autoreconf, libtool)
- MacOS X: XCode 4 or newer, doxygen, mactex, autotools (autoconf, autoheader, aclocal, automake, autoreconf, libtool)

Chapter 3

How to rebuild library

3.1 Building on Windows

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

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.

3.2 Building on debian-based linux systems

Requirement: 64-bit and 32-bit debian system, ubuntu Typical set of packages:

```
sudo apt-get install build-essential make cmake curl git ruby1.9.1 autotools-  
dev automake autoconf libtool doxygen bison flex debhelper lintian texlive texlive  
-latex-extra texlive-latex texlive-fonts-extra texlive-lang-cyrillic java-1.7.0-  
openjdk java-1.7.0-openjdk-devel default-jre-headless default-jdk openjdk-6-jdk  
rpm-build rpm-devel rpmlint pkg-config check dh-autoreconf hardening-wrapper  
libfl-dev lsb-release
```

For ARM cross-compiling install gcc-arm-linux-gnueabi from your ARM toolchain.

It's required to match library and host architecture: 32-bit library can be built only at 32-bit host, 64-bit library - only at 64-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 MacOS 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.

3.4 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.

3.5 Building on redhat-based linux systems

Requirement: 64-bit redhat-based system (Fedora, Red Hat, SUSE) Typical set of packages:

```
sudo apt-get install build-essential make cmake curl git ruby1.9.1 autotools-  
dev automake autoconf libtool doxygen bison flex debhelper lintian texlive texlive  
-latex-extra texlive-latex texlive-fonts-extra texlive-lang-cyrillic java-1.7.0-  
openjdk java-1.7.0-openjdk-devel default-jre-headless default-jdk openjdk-6-jdk  
rpm-build rpm-devel rpmlint pkg-config check dh-autoreconf hardening-wrapper  
libf1-dev lsb-release
```

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.

3.6 Source code access

The source codes of the libximc library can be found on [github](#).

Chapter 4

How to use with...

To acquire the first skills of using the library, a simple `testappeasy_C` test application has been created. Languages other than C are supported using calls with conversion of arguments of the `stdcall` type. A simple C test application is located in the `'examples/test_C'` directory, a C# project is located in `'examples/test_CSharp'`, on VB.NET - in `'examples/test_VBNET'`, for delphi 6 - in `'example/test_Delphi'`, for matlab - `'examples/test_MATLAB'`, for Java - `'examples/test_Java'`, for Python - `'examples/test_Python'`, for LabWindows - `'examples/test_LabWindows'`. Libraries, header files and other necessary files are located in the directories `'ximc/win32'`, `'ximc/win64'`, `'ximc/macosx'` and the like. The developer kit also includes already compiled examples: `testapp` and `testappeasy_x32` and `x64` bits for windows and only `x64` bits for macOS X, `test_CSharp`, `test_VBNET`, `test_Delphi` - only `x32` bits, `test_Java` - cross-platform, `test_MATLAB` and `test_Python` do not require compilation, `test_LabWindows` - 64-bit build is installed by default.

Note

SDK requires Microsoft Visual C++ Redistributable Package (provided with SDK - `vc_redist.x86` or `vc_redist.x64`)

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

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

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

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

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

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

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

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

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

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

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

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

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

Then compile test application:

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

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

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

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

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

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

```
make
```

In case of cross-compilation (target architecture differs from the current system architecture) feed -m64 or -m32 flag to compiler. On MacOS 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 MacOS 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).

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

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

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

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

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

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

In case of the 8Eth1 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).

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

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

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

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

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

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

Then rebuild sources:

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

or for Windows or MacOS X

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

Then build a jar:

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

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

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

To run:

```
python xxxx.py
```

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

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

Before launch:

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

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

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

On Windows before the start nothing needs to be done

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

```
testximc
```

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

4.1 Generic logging facility

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

4.2 Required permissions

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

4.3 C-profiles

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

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

4.4 Python-profiles

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

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

Chapter 5

Working with user units

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

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

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 user units.

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

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

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

5.3 Coordinate correction table for more accurate positioning

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

Deprecated.

Data Fields

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

6.1.1 Detailed Description

Deprecated.

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 (V).

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](#)
"Supply voltage of H-bridge's MOSFETs" raw data from ADC.
- unsigned int [ACurrent_ADC](#)
"Winding A current" raw data from ADC.
- unsigned int [BCurrent_ADC](#)
"Winding B current" raw data from ADC.
- unsigned int [FullCurrent_ADC](#)
"Full current" raw data from ADC.
- unsigned int [Temp_ADC](#)
Voltage from temperature sensor, raw data from ADC.
- unsigned int [Joy_ADC](#)
Joystick raw data from ADC.
- unsigned int [Pot_ADC](#)
Voltage on analog input, raw data from ADC.
- unsigned int [L5_ADC](#)
USB supply voltage after the current sense resistor, raw data from ADC.
- unsigned int [H5_ADC](#)
USB Power supply 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](#)
"Supply voltage on the top of H-bridge's MOSFETs" calibrated data (in tens of mV).
- int [ACurrent](#)
"Winding A current" calibrated data (in mA).
- int [BCurrent](#)
"Winding B current" calibrated data (in mA).
- int [FullCurrent](#)
"Full current" calibrated data (in mA).
- int [Temp](#)
Temperature, calibrated data (in tenths of degrees Celsius).
- int [Joy](#)
Joystick, calibrated data.
- int [Pot](#)
Analog input, calibrated data.
- int [L5](#)
USB supply voltage after the current sense resistor (in tens of mV).
- int [H5](#)
USB power supply (in tens of mV).
- unsigned int **deprecated**
- int [R](#)
Motor winding resistance in mOhms (is only used with stepper motors).
- int [L](#)
Motor winding pseudo inductance in uH (is only used with stepper motors).

6.2.1 Detailed Description

Analog data.

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

See Also

[get_analog_data](#)
[get_analog_data](#)

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

USB power supply (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 uH (is only used with stepper motors).

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, raw data 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 motors).

6.2.2.23 int SupVoltage

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

6.2.2.24 unsigned int SupVoltage_ADC

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

6.2.2.25 int Temp

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

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

6.3.1 Detailed Description

Brake settings.

This structure contains brake control parameters.

See Also

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

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 turning on motor power and turning off the brake.

6.3.2.3 unsigned int t2

Time in ms between the brake turning off 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 the brake turning on.

6.3.2.5 unsigned int t4

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

6.4 calibration_settings_t Struct Reference

Calibration settings.

Data Fields

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

6.4.1 Detailed Description

Calibration settings.

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

See Also

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

6.4.2 Field Documentation

6.4.2.1 float CSS1.A

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

6.4.2.2 float CSS1.B

Offset for the analog measurements of the A winding current.

6.4.2.3 float CSS2.A

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

6.4.2.4 float CSS2.B

Offset for the analog measurements of the B winding current.

6.4.2.5 float FullCurrent_A

Scaling factor for the analog measurements of the full current.

6.4.2.6 float FullCurrent_B

Offset for the analog measurements of the full current.

6.5 calibration_t Struct Reference

Calibration structure.

Data Fields

- double [A](#)
is a conversion factor which is equal number of millimeters (or other units) per one step
- unsigned int [MicrostepMode](#)
is a controller setting which is determine a step division mode

6.5.1 Detailed Description

Calibration structure.

6.6 chart_data_t Struct Reference

Additional device state.

Data Fields

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

- Analog input value, dimensionless.*
 - unsigned int [Joy](#)
The joystick position, dimensionless.
 - int [AveragedPowerRatio](#)
The ratio of motor supplied power to nominal motor power, as a percentage.

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 AveragedPowerRatio

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

6.6.2.2 unsigned int Joy

The joystick position, dimensionless.

Range: 0..10000

6.6.2.3 unsigned int Pot

Analog input value, dimensionless.

Range: 0..10000

6.6.2.4 int WindingCurrentA

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

6.6.2.5 int WindingCurrentB

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

6.6.2.6 int WindingCurrentC

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

6.6.2.7 int WindingVoltageA

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

6.6.2.8 int WindingVoltageB

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

6.6.2.9 int WindingVoltageC

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

6.7 control_settings_calb_t Struct Reference

User unit control settings.

Data Fields

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

6.7.1 Detailed Description

User unit control settings.

This structure contains control parameters.

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

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

See Also

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

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).

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

6.7.2.3 float MaxSpeed[10]

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

6.7.2.4 unsigned int Timeout[9]

Timeout[i] is timeout in ms.

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

6.8 control_settings_t Struct Reference

Control settings.

Data Fields

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

6.8.1 Detailed Description

Control settings.

This structure contains control parameters.

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

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

See Also

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

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).

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

6.8.2.3 unsigned int MaxSpeed[10]

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

Range: 0..100000.

6.8.2.4 unsigned int Timeout[9]

Timeout[i] is timeout in ms.

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

6.8.2.5 int uDeltaPosition

Fractional part of the shift in micro steps.

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

6.8.2.6 unsigned int uMaxSpeed[10]

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

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

6.9 controller_name_t Struct Reference

Controller name and settings flags.

Data Fields

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

6.9.1 Detailed Description

Controller name and settings flags.

See Also

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

6.9.2 Field Documentation

6.9.2.1 char ControllerName[17]

User controller name.

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

6.9.2.2 unsigned int CtrlFlags

[Flags of internal controller settings.](#)

6.10 ctp_settings_t Struct Reference

Control position settings (used with stepper motor only)

Data Fields

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

6.10.1 Detailed Description

Control position settings (used with stepper motor only)

When controlling the step motor with the encoder (CTP_BASE=0), it is possible to detect the loss of steps. The controller knows the number of steps per revolution (GENG::StepsPerRev) and the encoder resolution (GFBS::IPT). When the control is enabled (CTP_ENABLED is set), the controller stores the

current position in the steps of SM and the current position of the encoder. Next, the encoder position is converted into steps at each step, and if the difference between the current position in steps and the encoder position is greater than CTPMinError, the flag STATE_CTP_ERROR is set.

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

See Also

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

6.10.2 Field Documentation

6.10.2.1 unsigned int CTPFlags

[Position control flags](#).

6.10.2.2 unsigned int CTPMinError

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

Measured in steps.

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 the manufacturer.

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 the manufacturer.

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

Controller information structure.

Data Fields

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

6.13.1 Detailed Description

Controller information structure.

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

The minor number of the hardware version.

6.13.2.3 unsigned int Release

Release version.

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

User unit edges settings.

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

User unit edges settings.

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

See Also

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

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 may lead to device malfunction, which can cause irreversible damage to the board.

See Also

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

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).

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

6.16.2.6 int uRightBorder

Right border position in microsteps.

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

6.17 emf_settings_t Struct Reference

EMF settings.

Data Fields

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

6.17.1 Detailed Description

EMF settings.

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

See Also

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

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

Motor winding inductance.

6.17.2.4 float R

Motor winding resistance.

6.18 encoder_information_t Struct Reference

Deprecated.

Data Fields

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

6.18.1 Detailed Description

Deprecated.

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

Deprecated.

Data Fields

- float [MaxOperatingFrequency](#)
Maximum operation frequency (kHz).
- float [SupplyVoltageMin](#)

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

6.19.1 Detailed Description

Deprecated.

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

Maximum 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_advansed_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.

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.

In user units.

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

See Also

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

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

See Also

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

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 a device
<i>EngineType</i>	engine type
<i>DriverType</i>	driver type

See Also

[get_entype_settings](#), [set_entype_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 This data is stored in the controller's flash memory.

Data Fields

- unsigned int **Param1**

6.24.1 Detailed Description

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

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

See Also

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

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 events are signaled through a rising front, and output states are signaled by a high logic state.

See Also

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

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

Deprecated.

Data Fields

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

6.27.1 Detailed Description

Deprecated.

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

Deprecated.

Data Fields

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

6.28.1 Detailed Description

Deprecated.

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 ($g * cm^2$).

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

Maximum speed on the input shaft (rpm).

Data type: float.

6.28.2.5 float RatedInputTorque

Maximum 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.

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

A useful structure that contains position value in steps and microsteps for stepper motor and encoder steps for 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 the selected step division mode (see MicrostepMode field in engine_settings).

6.31 globally_unique_identifier_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.

Manufacturer only.

See Also

[get_globally_unique_identifier](#)

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

Deprecated.

Data Fields

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

6.32.1 Detailed Description

Deprecated.

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

Deprecated.

Data Fields

- float [MaxOperatingFrequency](#)
Maximum operation frequency (kHz).
- float [SupplyVoltageMin](#)

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

6.33.1 Detailed Description

Deprecated.

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

Maximum current consumption (mA).

Data type: float.

6.33.2.2 float MaxOperatingFrequency

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

6.35.1 Detailed Description

Position calibration settings.

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

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`

Fractional part of the speed for first motion, microsteps.

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

6.35.2.6 `int uHomeDelta`

Fractional part of the delta distance, microsteps.

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

6.35.2.7 `unsigned int uSlowHome`

Part of the speed for second motion, microsteps.

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

6.36 `init_random_t` Struct Reference

Random key.

Data Fields

- `uint8_t key` [16]
Random key.

6.36.1 Detailed Description

Random key.

Manufacturer only.

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

See Also

[get_init_random](#)

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 upper 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 falls outside the DeadZone limits, a movement begins. Speed is defined by the joystick position in the range of the DeadZone limit to the maximum deviation. Joystick positions inside the DeadZone limits correspond to zero speed (a soft stop of the motion), and positions beyond the Low and High limits correspond to 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 the next speed in the list is zero (both integer and microstep parts), the button press is ignored. The first speed in the list shouldn't be zero.

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

See Also

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

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 upper 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 considering the current step division mode) or encoder counts per second.
- int [Error](#) [25]
Current error in microsteps per second (whole steps are recalculated considering 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 stored 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 considering 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 considering the current step division mode) or encoder counts per second.

6.39 motor_information_t Struct Reference

Deprecated.

Data Fields

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

6.39.1 Detailed Description

Deprecated.

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

Deprecated.

Data Fields

- unsigned int [MotorType](#)
Motor Type flags.

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

6.40.1 Detailed Description

Deprecated.

Physical characteristics and limitations of the motor.

See Also

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

6.40.2 Field Documentation

6.40.2.1 float DetentTorque

Holding torque position with unpowered windings (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 motors (A).

Data type: float.

6.40.2.10 float NominalPower

Nominal power (W).

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

6.40.2.11 float NominalSpeed

Not used.

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

6.40.2.12 float NominalTorque

Nominal torque (mN * m).

Used for DC and BLDC engines. 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 motors.

6.40.2.16 float RotorInertia

Rotor inertia (g * cm²).

Data type: float.

6.40.2.17 float SpeedConstant

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

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 that determines the proportionality constant between the maximum rotor torque and current flowing in the winding (mN * m / A).

Used mainly for DC motors. Data type: float.

6.40.2.21 float WindingInductance

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

Data type: float.

6.40.2.22 float WindingResistance

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

Data type: float.

6.41 move_settings_calb_t Struct Reference

User units move settings.

Data Fields

- float [Speed](#)
Target speed.
- float [Accel](#)
Motor shaft acceleration, steps/s² (stepper motor) or RPM/s (DC).
- float [Decel](#)
Motor shaft deceleration, steps/s² (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

User units move settings.

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/s² (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, steps/s² (stepper motor) or 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, steps/s² (stepper motor) or RPM/s (DC).*
unsigned int [Decel](#)
- *Motor shaft deceleration, steps/s² (stepper motor) or RPM/s (DC).*
unsigned int [AntiplaySpeed](#)
- *Speed in antiplay mode, full steps/s (stepper motor) or RPM (DC).*
unsigned int [uAntiplaySpeed](#)
- *Speed in antiplay mode, microsteps/s.*
unsigned int [MoveFlags](#)
- *Flags of the motion parameters.*

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, steps/s² (stepper motor) or 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/s² (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.

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

6.42.2.7 unsigned int uSpeed

Target speed in microstep fractions/s.

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

6.43 network_settings_t Struct Reference

Network settings.

Data Fields

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

6.43.1 Detailed Description

Network settings.

Manufacturer only. This structure contains network settings.

See Also

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

6.43.2 Field Documentation

6.43.2.1 unsigned int DefaultGateway[4]

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

6.43.2.2 unsigned int DHCPEnabled

Indicates the method to get the IP-address.

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

6.43.2.3 unsigned int `IPv4Address`[4]

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

6.43.2.4 unsigned int `SubnetMask`[4]

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

6.44 `nonvolatile_memory_t` Struct Reference

Structure contains user data to save into the FRAM.

Data Fields

- unsigned int `UserData` [7]
User data.

6.44.1 Detailed Description

Structure contains user data to save into the FRAM.

See Also

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

6.44.2 Field Documentation

6.44.2.1 unsigned int `UserData`[7]

User data.

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

6.45 `password_settings_t` Struct Reference

The web-page user password.

Data Fields

- char `UserPassword` [21]
Password for the web-page that the user can change with a USB command or via web-page.

6.45.1 Detailed Description

The web-page user password.

Manufacturer only. This structure contains the user password.

See Also

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

6.45.2 Field Documentation

6.45.2.1 char UserPassword[21]

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

6.46 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.

6.46.1 Detailed Description

PID settings.

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

See Also

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

6.47 power_settings_t Struct Reference

Step motor power settings.

Data Fields

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

6.47.1 Detailed Description

Step motor power settings.

See Also

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

6.47.2 Field Documentation

6.47.2.1 unsigned int CurrentSetTime

Time in ms to reach the nominal current.

6.47.2.2 unsigned int CurrReductDelay

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

6.47.2.3 unsigned int HoldCurrent

Holding current, as percent of the nominal current.

Range: 0..100.

6.47.2.4 unsigned int PowerFlags

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

6.47.2.5 unsigned int PowerOffDelay

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

6.48 `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, in 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, in tens of mV.
- unsigned int [CriticalT](#)
Maximum temperature, which triggers ALARM state, in tenths of degrees Celsius.
- unsigned int [Criticalusb](#)
Maximum USB current which triggers ALARM state, in mA.
- unsigned int [CriticalUusb](#)
Maximum USB voltage which triggers ALARM state, in tens of mV.
- unsigned int [MinimumUusb](#)
Minimum USB voltage which triggers ALARM state, in tens of mV.
- unsigned int [Flags](#)
Flags of secure settings.

6.48.1 Detailed Description

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

These data are used for device testing and deep recalibration by the manufacturer only.

See Also

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

6.48.2 Field Documentation

6.48.2.1 unsigned int `Criticalpwr`

Maximum motor current which triggers ALARM state, in mA.

6.48.2.2 unsigned int `Criticalusb`

Maximum USB current which triggers ALARM state, in mA.

6.48.2.3 unsigned int `CriticalT`

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

6.48.2.4 unsigned int `CriticalUpwr`

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

6.48.2.5 unsigned int CriticalUusb

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

6.48.2.6 unsigned int Flags

[Flags of secure settings.](#)

6.48.2.7 unsigned int LowUpwrOff

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

6.48.2.8 unsigned int MinimumUusb

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

6.49 serial_number_t Struct Reference

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

Data Fields

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

6.49.1 Detailed Description

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

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

See Also

[set_serial_number](#)

6.49.2 Field Documentation

6.49.2.1 uint8_t Key[32]

Protection key (256 bit).

6.49.2.2 unsigned int Major

The major number of the hardware version.

6.49.2.3 unsigned int Minor

The minor number of the hardware version.

6.49.2.4 unsigned int Release

Number of edits this release of hardware.

6.49.2.5 unsigned int SN

New board serial number.

6.50 `set_position_calb_t` Struct Reference

User unit position information.

Data Fields

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

6.50.1 Detailed Description

User unit position information.

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

See Also

[set_position](#)

6.50.2 Field Documentation

6.50.2.1 long_t `EncPosition`

Encoder position.

6.50.2.2 unsigned int `PosFlags`

[Position setting flags.](#)

6.50.2.3 `float` Position

The position in the engine.

6.51 `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.

6.51.1 Detailed Description

Position information.

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

See Also

[set_position](#)

6.51.2 Field Documentation

6.51.2.1 `long_t` EncPosition

Encoder position.

6.51.2.2 `unsigned int` PosFlags

[Position setting flags.](#)

6.51.2.3 `int` uPosition

Microstep position is only used with stepper motors.

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

6.52 `stage_information_t` Struct Reference

Deprecated.

Data Fields

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

6.52.1 Detailed Description

Deprecated.

Stage information. Deprecated.

See Also

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

6.52.2 Field Documentation

6.52.2.1 char Manufacturer[17]

Manufacturer.

Max string length: 16 chars.

6.52.2.2 char PartNumber[25]

Series and PartNumber.

Max string length: 24 chars.

6.53 stage_name_t Struct Reference

Stage username.

Data Fields

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

6.53.1 Detailed Description

Stage username.

See Also

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

6.53.2 Field Documentation

6.53.2.1 char PositionerName[17]

User's positioner name.

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

6.54 stage_settings_t Struct Reference

Deprecated.

Data Fields

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

6.54.1 Detailed Description

Deprecated.

Stage settings.

See Also

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

6.54.2 Field Documentation

6.54.2.1 float HorizontalLoadCapacity

Horizontal load capacity (kg).

Data type: float.

6.54.2.2 float LeadScrewPitch

Lead screw pitch (mm).

Data type: float.

6.54.2.3 float MaxCurrentConsumption

Maximum current consumption (A).

Data type: float.

6.54.2.4 float MaxSpeed

Maximum speed (Units/c).

Data type: float.

6.54.2.5 float SupplyVoltageMax

Maximum supply voltage (V).

Data type: float.

6.54.2.6 float SupplyVoltageMin

Minimum supply voltage (V).

Data type: float.

6.54.2.7 float TravelRange

Travel range (Units).

Data type: float.

6.54.2.8 char Units[9]

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

Max string length: 8 chars.

6.54.2.9 float VerticalLoadCapacity

Vertical load capacity (kg).

Data type: float.

6.55 status_calb_t Struct Reference

User unit device's 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.
- 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, tenths of degrees Celsius.
- unsigned int [Flags](#)
Status flags.
- unsigned int [GPIOFlags](#)
Status flags of the GPIO outputs.
- unsigned int [CmdBufFreeSpace](#)
This field is a service field.

6.55.1 Detailed Description

User unit device's state.

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

See Also

`get_status_impl`

6.55.2 Field Documentation

6.55.2.1 unsigned int CmdBufFreeSpace

This field is a service field.

It shows the number of free synchronization chain buffer cells.

6.55.2.2 float CurPosition

Current position.

Corrected by the table.

6.55.2.3 float CurSpeed

Motor shaft speed.

6.55.2.4 int CurT

Temperature, tenths of degrees Celsius.

6.55.2.5 long_t EncPosition

Current encoder position.

6.55.2.6 unsigned int EncSts

[Encoder state.](#)

6.55.2.7 unsigned int Flags

[Status flags.](#)

6.55.2.8 unsigned int GPIOFlags

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

6.55.2.9 int Ipwr

Engine current, mA.

6.55.2.10 int Iusb

USB current, mA.

6.55.2.11 unsigned int MoveSts

[Flags of move state.](#)

6.55.2.12 unsigned int MvCmdSts

[Move command state.](#)

6.55.2.13 unsigned int PWRSts

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

6.55.2.14 int Upwr

Power supply voltage, tens of mV.

6.55.2.15 int Uusb

USB voltage, tens of mV.

6.55.2.16 unsigned int WindSts

[Winding state.](#)

6.56 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](#)
Fractional 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, tenths of degrees Celsius.

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

6.56.1 Detailed Description

Device state.

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

See Also

`get_status_impl`

6.56.2 Field Documentation

6.56.2.1 unsigned int CmdBufFreeSpace

This field is a service field.

It shows the number of free synchronization chain buffer cells.

6.56.2.2 int CurPosition

Current position.

6.56.2.3 int CurSpeed

Motor shaft speed in steps/s or rpm.

6.56.2.4 int CurT

Temperature, tenths of degrees Celsius.

6.56.2.5 long_t EncPosition

Current encoder position.

6.56.2.6 unsigned int EncSts

[Encoder state.](#)

6.56.2.7 unsigned int Flags

[Status flags.](#)

6.56.2.8 unsigned int GPIOFlags

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

6.56.2.9 int Ipwr

Engine current, mA.

6.56.2.10 int Iusb

USB current, mA.

6.56.2.11 unsigned int MoveSts

[Flags of move state.](#)

6.56.2.12 unsigned int MvCmdSts

[Move command state.](#)

6.56.2.13 unsigned int PWRSts

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

6.56.2.14 int uCurPosition

Step motor shaft position in microsteps.

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

6.56.2.15 int uCurSpeed

Fractional part of motor shaft speed in microsteps.

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

6.56.2.16 int Upwr

Power supply voltage, tens of mV.

6.56.2.17 int Uusb

USB voltage, tens of mV.

6.56.2.18 unsigned int WindSts

[Winding state.](#)

6.57 sync_in_settings_calb_t Struct Reference

User unit synchronization settings.

Data Fields

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

6.57.1 Detailed Description

User unit synchronization settings.

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

See Also

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

6.57.2 Field Documentation

6.57.2.1 unsigned int ClutterTime

Input synchronization pulse dead time (us).

6.57.2.2 float Position

Desired position or shift.

6.57.2.3 float Speed

Target speed.

6.57.2.4 unsigned int SyncInFlags

[Flags for synchronization input setup.](#)

6.58 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 (us).
- int [Position](#)
Desired position or shift (full steps)
- int [uPosition](#)
The fractional part of a position or shift in microsteps.
- unsigned int [Speed](#)
Target speed (for stepper motor: steps/s, for DC: rpm).
- unsigned int [uSpeed](#)
Target speed in microsteps/s.

6.58.1 Detailed Description

Synchronization settings.

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

See Also

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

6.58.2 Field Documentation

6.58.2.1 unsigned int ClutterTime

Input synchronization pulse dead time (us).

6.58.2.2 unsigned int Speed

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

Range: 0..100000.

6.58.2.3 unsigned int SyncInFlags

[Flags for synchronization input setup.](#)

6.58.2.4 int uPosition

The fractional part of a position or shift in microsteps.

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

6.58.2.5 unsigned int uSpeed

Target speed in microsteps/s.

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

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

6.59.1 Detailed Description

Synchronization settings which use user units.

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

See Also

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

6.59.2 Field Documentation

6.59.2.1 float Accuracy

This is the neighborhood around the target coordinates, every point in which is treated as the target position. Getting in these points cause the stop impulse.

6.59.2.2 unsigned int SyncOutFlags

[Flags of synchronization output.](#)

6.59.2.3 unsigned int SyncOutPeriod

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

6.59.2.4 unsigned int SyncOutPulseSteps

This value specifies the duration of output pulse.

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

6.60 sync_out_settings_t Struct Reference

Synchronization settings.

Data Fields

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

6.60.1 Detailed Description

Synchronization settings.

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

See Also

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

6.60.2 Field Documentation

6.60.2.1 unsigned int Accuracy

This is the neighborhood around the target coordinates, every point in which is treated as the target position. Getting in these points cause the stop impulse.

6.60.2.2 unsigned int SyncOutFlags

[Flags of synchronization output.](#)

6.60.2.3 unsigned int `SyncOutPeriod`

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

6.60.2.4 unsigned int `SyncOutPulseSteps`

This value specifies the duration of output pulse.

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

6.60.2.5 unsigned int `uAccuracy`

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

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

6.61 `uart_settings_t` Struct Reference

UART settings.

Data Fields

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

6.61.1 Detailed Description

UART settings.

This structure contains UART settings.

See Also

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

6.61.2 Field Documentation

6.61.2.1 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 structure.
- struct [device_network_information_t](#)
Device network information structure.
- struct [feedback_settings_t](#)
Feedback settings.
- struct [home_settings_t](#)
Position calibration settings.
- struct [home_settings_calb_t](#)
Position calibration settings which use user units.
- struct [move_settings_t](#)
Move settings.
- struct [move_settings_calb_t](#)
User units move settings.
- struct [engine_settings_t](#)
Movement limitations and settings related to the motor.
- struct [engine_settings_calb_t](#)
Movement limitations and settings, related to the motor.
- struct [entype_settings_t](#)
Engine type and driver type settings.
- struct [power_settings_t](#)
Step motor power settings.
- struct [secure_settings_t](#)
This structure contains raw analog data from ADC embedded on board.
- struct [edges_settings_t](#)
Edges settings.
- struct [edges_settings_calb_t](#)
User unit edges settings.
- struct [pid_settings_t](#)

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

- struct [status_calb_t](#)
User unit device's state.
- struct [measurements_t](#)
The buffer holds no more than 25 points.
- struct [chart_data_t](#)
Additional device state.
- struct [device_information_t](#)
Controller information structure.
- struct [serial_number_t](#)
The structure contains a new serial number, hardware version, and valid key.
- struct [analog_data_t](#)
Analog data.
- struct [debug_read_t](#)
Debug data.
- struct [debug_write_t](#)
Debug data.
- struct [stage_name_t](#)
Stage username.
- struct [stage_information_t](#)
Deprecated.
- struct [stage_settings_t](#)
Deprecated.
- struct [motor_information_t](#)
Deprecated.
- struct [motor_settings_t](#)
Deprecated.
- struct [encoder_information_t](#)
Deprecated.
- struct [encoder_settings_t](#)
Deprecated.
- struct [hallsensor_information_t](#)
Deprecated.
- struct [hallsensor_settings_t](#)
Deprecated.
- struct [gear_information_t](#)
Deprecated.
- struct [gear_settings_t](#)
Deprecated.
- struct [accessories_settings_t](#)
Deprecated.
- struct [init_random_t](#)
Random key.
- struct [globally_unique_identifier_t](#)
Globally unique identifier.

Macros

- #define `XIMC_API`
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_nodvice` `-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 an OS name is a XIMC device.
- #define `ENUMERATE_ALL_COM` `0x02`
Check all COM devices.
- #define `ENUMERATE_NETWORK` `0x04`
Check network devices.

Flags of move state

This is a bit mask for bitwise operations. Specify move states.

See Also

[get_status](#)
[status.t::MoveSts](#), [get_status_impl](#)

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

Flags of internal controller settings

This is a bit mask for bitwise operations.

See Also

[set_controller_name](#)
[get_controller_name](#)
[controller_name.t::CtrlFlags](#), [get_controller_name](#), [set_controller_name](#)

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

Flags of power state of stepper motor

This is a bit mask for bitwise operations. Specify power states.

See Also

[get_status](#)
[status.t::PWRSts](#), [get_status_impl](#)

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

Status flags

This is a bit mask for bitwise operations. Controller flags returned by device query. Contains boolean part of controller state. May be combined with bitwise OR.

See Also

[get_status](#)
[status.t::Flags](#), [get_status_impl](#)

- #define [STATE_CONTR](#) 0x000003F
Flags of controller states.
- #define [STATE_ERRC](#) 0x0000001
Command error encountered.
- #define [STATE_ERRD](#) 0x0000002
Data integrity error encountered.
- #define [STATE_ERRV](#) 0x0000004

- *Value error encountered.*
- #define [STATE_EEPROM_CONNECTED](#) 0x0000010
EEPROM with settings is connected.
- #define [STATE_IS_HOMED](#) 0x0000020
Calibration performed.
- #define [STATE_SECUR](#) 0x1B3FFC0
Security flags.
- #define [STATE_ALARM](#) 0x0000040
The controller is in an alarm state, indicating that something dangerous has happened.
- #define [STATE_CTP_ERROR](#) 0x0000080
Control position error (is only used with stepper motor).
- #define [STATE_POWER_OVERHEAT](#) 0x0000100
Power driver overheat.
- #define [STATE_CONTROLLER_OVERHEAT](#) 0x0000200
Controller overheat.
- #define [STATE_OVERLOAD_POWER_VOLTAGE](#) 0x0000400
Power voltage exceeds safe limit.
- #define [STATE_OVERLOAD_POWER_CURRENT](#) 0x0000800
Power current exceeds safe limit.
- #define [STATE_OVERLOAD_USB_VOLTAGE](#) 0x0001000
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 external EXTIO input signal.

Status flags of the GPIO outputs

This is a bit mask for bitwise operations. GPIO state flags returned by device query. Contains boolean part of controller state. May be combined with bitwise OR.

See Also

[get_status](#)

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

- #define [STATE_DIG_SIGNAL](#) 0xFFFF
Flags of digital signals.
- #define [STATE_RIGHT_EDGE](#) 0x0001
Engine stuck at the right edge.
- #define [STATE_LEFT_EDGE](#) 0x0002
Engine stuck at the left edge.
- #define [STATE_BUTTON_RIGHT](#) 0x0004
Button "right" state (1 if pressed).

- #define [STATE.BUTTON_LEFT](#) 0x0008
Button "left" state (1 if pressed).
- #define [STATE.GPIO_PINOUT](#) 0x0010
External GPIO works as out if the flag is set; otherwise, it works as in.
- #define [STATE.GPIO_LEVEL](#) 0x0020
State of external GPIO pin.
- #define [STATE.BRAKE](#) 0x0200
State of Brake pin.
- #define [STATE.REV.SENSOR](#) 0x0400
State of Revolution sensor pin.
- #define [STATE.SYNC.INPUT](#) 0x0800
State of Sync input pin.
- #define [STATE.SYNC.OUTPUT](#) 0x1000
State of Sync output pin.
- #define [STATE.ENC.A](#) 0x2000
State of encoder A pin.
- #define [STATE.ENC.B](#) 0x4000
State of encoder B pin.

Encoder state

This is a bit mask for bitwise operations. Encoder state returned by device query.

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 has finished with an error, 0 - move command has finished correctly).
- #define [MVCMD_RUNNING](#) 0x80
Move command state (0 - move command has finished, 1 - move command is being executed).

Flags of the motion parameters

This is a bit mask for bitwise operations. Specify the motor shaft movement algorithm and list of limitations. Flags returned by the query of [get_move_settings](#).

See Also

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

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

Flags of engine settings

This is a bit mask for bitwise operations. Specify the motor shaft movement algorithm and list of limitations. Flags returned by query of engine settings. May be combined with bitwise OR.

See Also

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

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

Flags of microstep mode

This is a bit mask for bitwise operations. Specify settings for microstep mode. Used with step motors. Flags returned by query of engine settings. May be combined with bitwise OR

See Also

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

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

Flags of engine type

This is a bit mask for bitwise operations. Specify motor type. Flags returned by query of engine settings.

See Also

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

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

Flags of driver type

This is a bit mask for bitwise operations. Specify driver type. Flags returned by query of engine settings.

See Also

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

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

Flags of secure settings

This is a bit mask for bitwise operations. Flags returned by query of engine settings. Specify secure settings. Flags returned by query of secure settings.

See Also

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

- #define [ALARM_ON_DRIVER_OVERHEATING](#) 0x01
If this flag is set, enter the alarm state on the driver overheat signal.
- #define [LOW_UPWR_PROTECTION](#) 0x02
If this flag is set, turn off the motor when the voltage is lower than LowUpwrOff.
- #define [H_BRIDGE_ALERT](#) 0x04
If this flag is set then turn off the power unit with a signal problem in one of the transistor bridge.
- #define [ALARM_ON_BORDERS_SWAP_MISSET](#) 0x08
If this flag is set, enter Alarm state on borders swap misset.
- #define [ALARM_FLAGS_STICKING](#) 0x10
If this flag is set, only a STOP command can turn all alarms to 0.
- #define [USB_BREAK_RECONNECT](#) 0x20
If this flag is set, the USB brake reconnect module will be enabled.
- #define [ALARM_WINDING_MISMATCH](#) 0x40
If this flag is set, enter Alarm state when windings mismatch.
- #define [ALARM_ENGINE_RESPONSE](#) 0x80
If this flag is set, enter the Alarm state on response of the engine control action.

Position setting flags

This is a bit mask for bitwise operations. Flags used in setting position.

See Also

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

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

Feedback type.

This is a bit mask for bitwise operations.

See Also

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

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

Describes feedback flags.

This is a bit mask for bitwise operations.

See Also

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

- #define **FEEDBACK_ENC_REVERSE** 0x01
Reverse count of encoder.
- #define **FEEDBACK_ENC_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 going to the position specified in Position and uPosition if this flag is set.

Flags of synchronization output

This is a bit mask for bitwise operations.

See Also

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

- #define **SYNCOUT_ENABLED** 0x01
The synchronization out pin follows the synchronization logic if the flag is set.
- #define **SYNCOUT_STATE** 0x02
When the output state is fixed by the negative SYNCOUT_ENABLED flag, the pin state is in accordance with this flag state.
- #define **SYNCOUT_INVERT** 0x04
The low level is active if the flag is set.
- #define **SYNCOUT_IN_STEPS** 0x08
Use motor steps or encoder pulses instead of milliseconds for output pulse generation if the flag is set.
- #define **SYNCOUT_ONSTART** 0x10
Generate a synchronization pulse when movement starts.
- #define **SYNCOUT_ONSTOP** 0x20
Generate a synchronization pulse when movement stops.
- #define **SYNCOUT_ONPERIOD** 0x40
Generate a synchronization pulse every SyncOutPeriod encoder pulses.

External IO setup flags

This is a bit mask for bitwise operations.

See Also

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

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

External IO mode flags

This is a bit mask for bitwise operations.

See Also

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

- #define [EXTIO_SETUP_MODE_IN_BITS](#) 0x0F
Bits of the behavior selector when the signal on input goes to the active state.
- #define [EXTIO_SETUP_MODE_IN_NOP](#) 0x00
Do nothing.
- #define [EXTIO_SETUP_MODE_IN_STOP](#) 0x01
Issue STOP command, ceasing the engine movement.
- #define [EXTIO_SETUP_MODE_IN_PWOF](#) 0x02
Issue PWOF command, powering off all engine windings.
- #define [EXTIO_SETUP_MODE_IN_MOVR](#) 0x03
Issue MOVR command with last used settings.
- #define [EXTIO_SETUP_MODE_IN_HOME](#) 0x04
Issue HOME command.
- #define [EXTIO_SETUP_MODE_IN_ALARM](#) 0x05
Set Alarm when the signal goes to the active state.
- #define [EXTIO_SETUP_MODE_OUT_BITS](#) 0xF0
Bits of the output behavior selection.
- #define [EXTIO_SETUP_MODE_OUT_OFF](#) 0x00
EXTIO pin always set in inactive state.
- #define [EXTIO_SETUP_MODE_OUT_ON](#) 0x10
EXTIO pin always set in active state.
- #define [EXTIO_SETUP_MODE_OUT_MOVING](#) 0x20
EXTIO pin stays active during moving state.
- #define [EXTIO_SETUP_MODE_OUT_ALARM](#) 0x30
EXTIO pin stays active during the alarm state.
- #define [EXTIO_SETUP_MODE_OUT_MOTOR_ON](#) 0x40
EXTIO pin stays active when windings are powered.

Border flags

This is a bit mask for bitwise operations. Specify types of borders and motor behavior on borders. May be combined with bitwise OR.

See Also

[get_edges_settings](#)
[set_edges_settings](#)
[edges_settings.t::BorderFlags](#), [get_edges_settings](#), [set_edges_settings](#)

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

Limit switches flags

This is a bit mask for bitwise operations. Specify electrical behavior of limit switches like order and pulled positions. May be combined with bitwise OR.

See Also

[get_edges_settings](#)
[set_edges_settings](#)
[edges_settings.t::EnderFlags](#), [get_edges_settings](#), [set_edges_settings](#)

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

Brake settings flags

This is a bit mask for bitwise operations. Specify behavior of brake. May be combined with bitwise OR.

See Also

[get_brake_settings](#)
[set_brake_settings](#)
[brake_settings.t::BrakeFlags](#), [get_brake_settings](#), [set_brake_settings](#)

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

Control flags

This is a bit mask for bitwise operations. Specify motor control settings by joystick or buttons. May be combined with bitwise OR.

See Also

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

- #define **CONTROL_MODE_BITS** 0x03
Bits to control the engine by joystick or buttons.

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

Joystick flags

This is a bit mask for bitwise operations. Control joystick states.

See Also

[set_joystick_settings](#)
[get_joystick_settings](#)
[joystick_settings_t::JoyFlags](#), [get_joystick_settings](#), [set_joystick_settings](#)

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

Position control flags

This is a bit mask for bitwise operations. Specify control position settings. May be combined with bitwise OR.

See Also

[get_ctp_settings](#)
[set_ctp_settings](#)
[ctp_settings_t::CTPFlags](#), [get_ctp_settings](#), [set_ctp_settings](#)

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

Home settings flags

This is a bit mask for bitwise operations. Specify home command behavior. May be combined with bitwise OR.

See Also

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

- #define [HOME_DIR_FIRST](#) 0x001
The flag defines the direction of the 1st motion after execution of the home command.
- #define [HOME_DIR_SECOND](#) 0x002

- The flag defines the direction of the 2nd motion.*

 - #define [HOME_MV_SEC_EN](#) 0x004

Use the second phase of calibration to the home position, if set; otherwise the second phase is skipped.
- #define [HOME_HALF_MV](#) 0x008

If the flag is set, the stop signals are ignored during the first half-turn of the second movement.
- #define [HOME_STOP_FIRST_BITS](#) 0x030

Bits of the first stop selector.
- #define [HOME_STOP_FIRST_REV](#) 0x010

First motion stops by revolution sensor.
- #define [HOME_STOP_FIRST_SYN](#) 0x020

First motion stops by synchronization input.
- #define [HOME_STOP_FIRST_LIM](#) 0x030

First motion stops by limit switch.
- #define [HOME_STOP_SECOND_BITS](#) 0x0C0

Bits of the second stop selector.
- #define [HOME_STOP_SECOND_REV](#) 0x040

Second motion stops by revolution sensor.
- #define [HOME_STOP_SECOND_SYN](#) 0x080

Second motion stops by synchronization input.
- #define [HOME_STOP_SECOND_LIM](#) 0x0C0

Second motion stops by limit switch.
- #define [HOME_USE_FAST](#) 0x100

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

UART parity flags

This is a bit mask for bitwise operations.

See Also

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

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

Motor Type flags

This is a bit mask for bitwise operations.

See Also

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

- #define [MOTOR_TYPE_UNKNOWN](#) 0x00
- Unknown type of engine.*
- #define [MOTOR_TYPE_STEP](#) 0x01
- Step engine.*

- #define [MOTOR_TYPE_DC](#) 0x02
DC engine.
- #define [MOTOR_TYPE_BLDC](#) 0x03
BLDC engine.

Encoder settings flags

This is a bit mask for bitwise operations.

See Also

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

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

Magnetic brake settings flags

This is a bit mask for bitwise operations.

See Also

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

- #define [MB_AVAILABLE](#) 0x01
If the flag is set, the magnetic brake is available.
- #define [MB_POWERED_HOLD](#) 0x02
If this flag is set, the magnetic brake is on when powered.

Temperature sensor settings flags

This is a bit mask for bitwise operations.

See Also

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

- #define [TS_TYPE_BITS](#) 0x07
Bits of the temperature sensor type.
- #define [TS_TYPE_UNKNOWN](#) 0x00
Unknown type of sensor.
- #define [TS_TYPE_THERMOCOUPLE](#) 0x01
Thermocouple.
- #define [TS_TYPE_SEMICONDUCTOR](#) 0x02
The semiconductor temperature sensor.
- #define [TS_AVAILABLE](#) 0x08
If the flag is set, the temperature sensor is available.
- #define [LS_ON_SW1_AVAILABLE](#) 0x01
If the flag is set, the limit switch connected to pin SW1 is available.
- #define [LS_ON_SW2_AVAILABLE](#) 0x02
If the flag is set, the limit switch connected to pin SW2 is available.

- `#define LS_SW1_ACTIVE_LOW 0x04`
If the flag is set, the limit switch connected to pin SW1 is triggered by a low level on the pin.
- `#define LS_SW2_ACTIVE_LOW 0x08`
If the flag is set, the limit switch connected to pin SW2 is triggered by a low level on pin.
- `#define LS_SHORTED 0x10`
If the flag is set, the limit switches are shorted.

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

This is a bit mask for bitwise operations.

See Also

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

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

Typedefs

- `typedef unsigned long long ulong_t`
- `typedef long long long_t`
- `typedef int device_t`
Type describes device identifier.
- `typedef int result_t`
Type specifies result of any operation.
- `typedef uint32_t device_enumeration_t`
Type describes device enumeration structure.
- `typedef struct calibration_t calibration_t`
Calibration structure.
- `typedef struct device_network_information_t device_network_information_t`
Device network information structure.

Functions

Controller settings setup

Read and write functions for almost all controller settings.

- `result_t XIMC_API set_feedback_settings (device_t id, const feedback_settings_t *feedback_settings)`
Feedback settings.
- `result_t XIMC_API get_feedback_settings (device_t id, feedback_settings_t *feedback_settings)`
Feedback settings.
- `result_t XIMC_API set_home_settings (device_t id, const home_settings_t *home_settings)`
Set home settings.
- `result_t XIMC_API set_home_settings_calb (device_t id, const home_settings_calb_t *home_settings_calb, const calibration_t *calibration)`
Set user unit home settings.

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

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

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

Group of commands movement control

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

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

Group of save settings and load settings commands

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

Service commands

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

A group of EEPROM commands

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

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

Boards and drivers control

Functions for searching and opening/closing devices

- typedef char * [pchar](#)
Nevermind.
- typedef void([XIMC_CALLCONV](#) * [logging_callback_t](#))(int loglevel, const wchar_t *message, void *user_data)
Logging callback prototype.
- [device_t XIMC_API open_device](#) (const char *uri)
Open a device with OS uri and return identifier of the device which can be used in calls.
- [result_t XIMC_API close_device](#) ([device_t](#) *id)
Close specified device.
- [result_t XIMC_API 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)
Deprecated.
- [device_enumeration_t XIMC_API enumerate_devices](#) (int enumerate_flags, const char *hints)
Enumerate all XIMC-compatible devices.
- [result_t XIMC_API free_enumerate_devices](#) ([device_enumeration_t](#) device_enumeration)
Free memory returned by enumerate_devices.
- int [XIMC_API get_device_count](#) ([device_enumeration_t](#) device_enumeration)
Get device count.
- [pchar XIMC_API get_device_name](#) ([device_enumeration_t](#) device_enumeration, int device_index)
Get device name from the device enumeration.

- [result_t XIMC_API get_enumerate_device_serial](#) ([device_enumeration_t](#) device_enumeration, int device_index, [uint32_t](#) *serial)
Get device serial number from the device enumeration.
- [result_t XIMC_API get_enumerate_device_information](#) ([device_enumeration_t](#) device_enumeration, int device_index, [device_information_t](#) *device_information)
Get device information from the device enumeration.
- [result_t XIMC_API get_enumerate_device_controller_name](#) ([device_enumeration_t](#) device_enumeration, int device_index, [controller_name_t](#) *controller_name)
Get controller name from the device enumeration.
- [result_t XIMC_API get_enumerate_device_stage_name](#) ([device_enumeration_t](#) device_enumeration, int device_index, [stage_name_t](#) *stage_name)
Get stage name from the device enumeration.
- [result_t XIMC_API get_enumerate_device_network_information](#) ([device_enumeration_t](#) device_enumeration, int device_index, [device_network_information_t](#) *device_network_information)
Get device network information from the device enumeration.
- [result_t XIMC_API reset_locks](#) ()
Resets the error of incorrect data transmission.
- [result_t XIMC_API ximc_fix_usbser_sys](#) (const char *device_uri)
(Deprecated) 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 the alarm state on the 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 are placed on limit switches, if not set.

7.1.2.6 `#define BORDER_STOP_LEFT 0x02`

The motor should stop on the 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 the motor when 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 the stepper motor power off if this flag is set.

7.1.2.11 `#define CONTROL_BTN_LEFT_PUSHED_OPEN 0x04`

Pushed left button corresponds to the 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 the 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 the flag is set.

7.1.2.18 `#define CTP_BASE 0x02`

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

7.1.2.19 `#define CTP_ENABLED 0x01`

The position control is enabled if the flag is set.

7.1.2.20 `#define CTP_ERROR_CORRECTION 0x10`

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

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

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

7.1.2.35 `#define ENGINE_CURRENT_AS_RMS 0x02`

Engine current meaning flag.

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

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, the engine uses the maximum speed achievable with the present engine settings as its 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 an OS name is a XIMC device.

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

7.1.2.48 `#define EXTIO_SETUP_INVERT 0x02`

Interpret EXTIO state inverted if the flag is set.

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

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 behavior 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 the alarm state.

7.1.2.57 `#define EXTIO_SETUP_MODE_OUT_BITS 0xF0`

Bits of the output behavior 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 the 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`

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

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

7.1.2.74 `#define HOME_DIR_SECOND 0x002`

The flag defines the direction of the 2nd motion.

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

7.1.2.75 `#define HOME_HALF_MV 0x008`

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

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.

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

7.1.2.87 `#define LOW_UPWR_PROTECTION 0x02`

If this flag is set, turn off the motor when the 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 the controller is trying to move the motor.

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

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 has finished with an error, 0 - move command has finished correctly).

This flag makes sense when `MVCMD_RUNNING` signals movement completion.

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 rigt.

7.1.2.108 `#define MVCMD_RUNNING 0x80`

Move command state (0 - move command has 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 is enabled after PowerOffDelay if this flag is set.

7.1.2.113 `#define POWER_REDUCT_ENABLED 0x01`

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

7.1.2.114 `#define POWER_SMOOTH_CURRENT 0x04`

Current ramp-up/down are 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 the 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`

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

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

7.1.2.121 `#define RPM.DIV_1000 0x01`

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

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

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`

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

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

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).

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

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.

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

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.

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

7.1.2.137 `#define STATE_ERRC 0x0000001`

Command error encountered.

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

7.1.2.138 `#define STATE_ERRD 0x0000002`

Data integrity error encountered.

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

7.1.2.139 `#define STATE_ERRV 0x0000004`

Value error encountered.

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

7.1.2.140 `#define STATE_EXTIO_ALARM 0x1000000`

The error is caused by the external EXTIO 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 the flag is set; otherwise, it works as in.

7.1.2.143 `#define STATE_IS_HOMED 0x0000020`

Calibration performed.

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

7.1.2.144 `#define STATE_LEFT_EDGE 0x0002`

Engine stuck at the left edge.

7.1.2.145 `#define STATE_LOW_USB_VOLTAGE 0x0002000`

USB voltage is insufficient for normal operation.

7.1.2.146 `#define STATE_OVERLOAD_POWER_CURRENT 0x0000800`

Power current exceeds safe limit.

7.1.2.147 `#define STATE_OVERLOAD_POWER_VOLTAGE 0x0000400`

Power voltage exceeds safe limit.

7.1.2.148 `#define STATE_OVERLOAD_USB_CURRENT 0x0004000`

USB current exceeds safe limit.

7.1.2.149 `#define STATE_OVERLOAD_USB_VOLTAGE 0x0001000`

USB voltage exceeds safe limit.

7.1.2.150 `#define STATE_POWER_OVERHEAT 0x0000100`

Power driver overheat.

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

7.1.2.151 `#define STATE_REV_SENSOR 0x0400`

State of Revolution sensor pin.

7.1.2.152 `#define STATE_RIGHT_EDGE 0x0001`

Engine stuck at the right edge.

7.1.2.153 `#define STATE_SECUR 0x1B3FFC0`

Security flags.

7.1.2.154 `#define STATE_SYNC_INPUT 0x0800`

State of Sync input pin.

7.1.2.155 `#define STATE_SYNC_OUTPUT 0x1000`

State of Sync output pin.

7.1.2.156 `#define STATE_WINDING_RES_MISMATCH 0x0100000`

The difference between winding resistances is too large.

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

7.1.2.157 `#define SYNCIN_ENABLED 0x01`

Synchronization in mode is enabled if this flag is set.

7.1.2.158 `#define SYNCIN_GOTOPOSITION 0x04`

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

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

7.1.2.159 `#define SYNCIN_INVERT 0x02`

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

7.1.2.160 `#define SYNCOUT_ENABLED 0x01`

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

Otherwise, it is governed by the SYNCOUT_STATE flag.

7.1.2.161 `#define SYNCOUT_IN_STEPS 0x08`

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

7.1.2.162 `#define SYNCOUT_INVERT 0x04`

The low level is active if the flag is set.

Otherwise, the high level is active.

7.1.2.163 `#define SYNCOUT_ONPERIOD 0x40`

Generate a synchronization pulse every SyncOutPeriod encoder pulses.

7.1.2.164 `#define SYNCOUT_ONSTART 0x10`

Generate a synchronization pulse when movement starts.

7.1.2.165 `#define SYNCOUT_ONSTOP 0x20`

Generate a synchronization pulse when movement stops.

7.1.2.166 `#define SYNCOUT_STATE 0x02`

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

7.1.2.167 `#define UART_PARITY_BITS 0x03`

Bits of the parity.

7.1.2.168 `#define WIND_A_STATE_ABSENT 0x00`

Winding A is disconnected.

7.1.2.169 `#define WIND_A_STATE_MALFUNC 0x02`

Winding A is short-circuited.

7.1.2.170 `#define WIND_A_STATE_OK 0x03`

Winding A is connected and working properly.

7.1.2.171 `#define WIND_A_STATE_UNKNOWN 0x01`

Winding A state is unknown.

7.1.2.172 `#define WIND_B_STATE_ABSENT 0x00`

Winding B is disconnected.

7.1.2.173 `#define WIND_B_STATE_MALFUNC 0x20`

Winding B is short-circuited.

7.1.2.174 `#define WIND_B_STATE_OK 0x30`

Winding B is connected and working properly.

7.1.2.175 `#define WIND_B_STATE_UNKNOWN 0x10`

Winding B state is unknown.

7.1.2.176 `#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 the stage's EEPROM to the controller's RAM.

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

Parameters

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

7.1.4.4 **result_t XIMC_API** command_eesave_settings (**device_t** id)

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

Can be used by the manufacturer only.

Parameters

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

7.1.4.5 **result_t XIMC_API** command_home (**device_t** id)

Moving to home position.

Moving algorithm:

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

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

Moving settings can be set by set_home_settings/set_home_settings_calb.

Parameters

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

See Also

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

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
<i>out</i>	<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 continuous moving to the left.

Parameters

<i>id</i>	An identifier of a 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 position to a distance Antiplay defined in engine settings.

Then moves to the initial position.

Parameters

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

7.1.4.9 **result_t XIMC_API** command_move (**device_t** id, int Position, int uPosition)

Move to position.

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

Parameters

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

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

Move to position using user units.

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

Parameters

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

Note

The parameter Position is adjusted by the correction table.

7.1.4.11 **result_t XIMC_API** command_movr (**device_t** id, int DeltaPosition, int uDeltaPosition)

Shift by a set offset.

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

Parameters

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

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

Shift by a set offset using user units.

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

Parameters

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

Note

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

7.1.4.13 **result_t XIMC_API** command_power_off (**device_t** id)

Immediately power off the motor regardless its state.

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

Parameters

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

See Also

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

7.1.4.14 **result_t XIMC_API** command_read_robust_settings (**device_t** id)

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

Manufacturer only.

Parameters

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

7.1.4.15 **result_t XIMC_API** command_read_settings (**device_t** id)

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

Parameters

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

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 a device
-----------	---------------------------

7.1.4.18 **result_t XIMC_API** command_save_robust_settings (**device_t** id)

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

Manufacturer only.

Parameters

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

7.1.4.19 **result_t XIMC_API** command_save_settings (**device_t** id)

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

Parameters

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

7.1.4.20 **result_t XIMC_API** command_sstp (**device_t** id)

Soft stop the engine.

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

Parameters

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

7.1.4.21 **result_t XIMC_API** command_start_measurements (**device_t** id)

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

Parameters

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

7.1.4.22 **result_t XIMC_API** command_stop (**device_t** id)

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

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

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

Parameters

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

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

Update firmware.

Manufacturer only. Service command

Parameters

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

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.
out	<i>ret</i>	RESULT_OK if controller has stopped and result of the first get_status command which returned anything other than RESULT_OK otherwise.

7.1.4.25 **result_t XIMC_API** command_zero (**device_t** id)

Sets the current position to 0.

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

Parameters

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

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

Enumerate all XIMC-compatible devices.

Parameters

in	<i>enumerate_flags</i>	enumerate devices flags
in	<i>hints</i>	extended search information

hints is a string of form "key=value \n key2=value2". *Unrecognized key-value pairs are ignored.* Key list: addr (required!) - mandatory flag used together with the 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. Example: "addr=192.168.1.1,172.16.2.3". Absent value means broadcast discovery. Example: "addr=". 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. Example: "addr= \n adapter_addr=192.168.0.100".

7.1.4.27 **result_t XIMC_API** free_enumerate_devices (**device_enumeration_t** device_enumeration)

Free memory returned by *enumerate_devices*.

Parameters

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

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

Deprecated.

Read additional accessory information from the EEPROM.

Parameters

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

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

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

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

Parameters

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

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

Read the controller's bootloader version.

Parameters

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

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

Read break control settings.

Parameters

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

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

Read calibration settings.

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

See Also

[calibration_settings_t](#)

Parameters

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

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

See Also

[chart_data_t](#)

Parameters

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

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

Read motor control settings.

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

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

Parameters

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

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)

Set calibrated motor control settings.

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

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

Parameters

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

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

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

Parameters

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

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

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

When controlling the step motor with an encoder (CTP_BASE=0), it is possible to detect the loss of steps. The controller knows

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

Parameters

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

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.

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

Parameters

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

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_-</i> <i>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_-</i> <i>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 a device
out	<i>edges_settings</i>	edges settings, types of borders, motor behavior and electrical behavior 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 in user units.

See Also

[set_edges_settings_calb](#)

Parameters

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

Note

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

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 a 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)`

Deprecated.

Read encoder information from the EEPROM.

Parameters

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

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

Deprecated.

Read encoder settings from the EEPROM.

Parameters

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

7.1.4.47 **result_t XIMC_API** `get_engine_advansed_setup (device_t id, engine_advansed_setup_t * engine_advansed_setup)`

Read engine advanced settings.

See Also

[set_engine_advansed_setup](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>engine_</i> <i>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 reads the structure containing a set of useful motor settings stored in the controller's memory. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics.

See Also

[set_engine_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>engine_settings</i>	engine settings

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 user unit engine settings.

This function reads the structure containing a set of useful motor settings stored in the controller's memory. These settings specify the motor shaft movement algorithm, list of limitations and rated characteristics.

See Also

[set_engine_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>engine_settings-</i> <i>_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 a device
out	<i>entype_settings</i>	structure contains motor type and power driver type settings

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.

Currently, it is not in use.

See Also

[set_extended_settings](#)

Parameters

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

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 the controller's memory.

See Also

[set_extio_settings](#)

Parameters

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

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 a device
out	<i>IPS</i>	number of encoder counts per shaft revolution. Range: 1..65535. The field is obsolete, it is recommended to write 0 to IPS and use the extended CountsPerTurn field. You may need to update the controller firmware to the latest version.
out	<i>FeedbackType</i>	type of feedback
out	<i>FeedbackFlags</i>	flags of feedback
out	<i>CountsPerTurn</i>	number of encoder counts per shaft revolution. Range: 1..4294967295. To use the CountsPerTurn field, write 0 in the IPS field, otherwise the value from the IPS field will be used.

7.1.4.59 **result_t XIMC_API** get_firmware_version (**device_t** id, unsigned int * Major, unsigned int * Minor, unsigned int * Release)

Read the controller's firmware version.

Parameters

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

7.1.4.60 **result_t XIMC_API** get_gear_information (**device_t** id, **gear_information_t** * gear_information)

Deprecated.

Read gear information from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>gear_information</i>	structure contains information about step gearhead

7.1.4.61 **result_t XIMC_API** get_gear_settings (**device_t** id, **gear_settings_t** * gear_settings)

Deprecated.

Read gear settings from the EEPROM.

Parameters

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

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 device, but is not a random value.

Manufacturer only. This unique device identifier can be used to initiate secure boot processes or as a serial number for USB or other end applications.

Parameters

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

7.1.4.63 **result_t XIMC_API** get_hallsensor_information (**device_t** id, **hallsensor_information_t** * hallsensor_information)

Deprecated.

Read hall sensor information from the EEPROM.

Parameters

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

7.1.4.64 **result_t XIMC_API** get_hallsensor_settings (**device_t** id, **hallsensor_settings_t** * hallsensor_settings)

Deprecated.

Read hall sensor settings from the EEPROM.

Parameters

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

7.1.4.65 **result_t XIMC_API** get_home_settings (**device_t** id, **home_settings_t** * home_settings)

Read home settings.

This function reads the structure with home position settings.

See Also

[home_settings_t](#)

Parameters

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

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 user unit home settings.

This function reads the structure with home position settings.

See Also

[home_settings_calb_t](#)

Parameters

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

7.1.4.67 **result_t XIMC_API** get_init_random (**device_t** id, **init_random_t** * init_random)

Read a random number from the controller.

Manufacturer only.

Parameters

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

7.1.4.68 **result_t XIMC_API** get_joystick_settings (**device_t** id, **joystick_settings_t** * joystick_settings)

Read joystick settings.

If joystick position falls outside DeadZone limits, a movement begins. The speed is defined by the joystick's position in the range of the DeadZone limit to the maximum deviation. Joystick positions inside DeadZone limits correspond to zero speed (a soft stop of the motion), and positions beyond Low and High limits correspond to MaxSpeed[i] or -MaxSpeed[i] (see command SCTL), where i = 0 by default and can be changed with the left/right buttons (see command SCTL). If the next speed in the list is zero (both integer and microstep parts), the button press is ignored. The first speed in the list shouldn't be zero. See the Joystick control section on https://doc.xisupport.com/en/8smc5-usb/8SMCn-USB/Technical-specification/Additional_features/Joystick_control.html for more information.

Parameters

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

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 speed error graph.

Filling the buffer starts with the command "start_measurements". The buffer holds 25 points; the points are taken with a period of 1 ms. To create a robust system, read data every 20 ms. If the buffer is full, it is recommended to repeat the readings every 5 ms until the buffer again becomes filled with 20 points.

To stop measurements just stop reading data. After buffer overflow measurements will stop automatically.

See Also

[measurements_t](#)

Parameters

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

7.1.4.70 **result_t XIMC_API** get_motor_information (**device_t** id, **motor_information_t** * motor_information)

Deprecated.

Read motor information from the EEPROM.

Parameters

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

7.1.4.71 **result_t XIMC_API** get_motor_settings (**device_t** id, **motor_settings_t** * motor_settings)

Deprecated.

Read motor settings from the EEPROM.

Parameters

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

7.1.4.72 **result_t XIMC_API** get_move_settings (**device_t** id, **move_settings_t** * move_settings)

Movement settings read command (speed, acceleration, threshold, etc.).

Parameters

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

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)

User unit movement settings read command (speed, acceleration, threshold, etc.).

Parameters

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

	<i>calibration</i>	user unit settings
--	--------------------	--------------------

7.1.4.74 **result_t XIMC_API** get_network_settings (**device_t** id, **network_settings_t** * network_settings)

Read network settings.

Manufacturer only. This function returns the current network settings.

See Also

net_settings_t

Parameters

<i>DHCPEnabled</i>	DHCP enabled (1) or not (0)
<i>IPv4Address[4]</i>	Array[4] with an IP address
<i>SubnetMask[4]</i>	Array[4] with a subnet mask address
<i>Default-Gateway[4]</i>	Array[4] with a default gateway address

7.1.4.75 **result_t XIMC_API** get_nonvolatile_memory (**device_t** id, **nonvolatile_memory_t** * nonvolatile_memory)

Read user data from FRAM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>nonvolatile-memory</i>	structure contains previously set user data.

7.1.4.76 **result_t XIMC_API** get_password_settings (**device_t** id, **password_settings_t** * password_settings)

Read the password.

Manufacturer only. This function reads the user password for the device's web-page.

See Also

pwd_settings_t

Parameters

<i>User-Password[20]</i>	Password for web-page
--------------------------	-----------------------

7.1.4.77 **result_t XIMC_API** get_pid_settings (**device_t** id, **pid_settings_t** * pid_settings)

Read PID settings.

This function reads the structure containing a set of motor PID settings stored in the controller's memory. These settings specify the behavior of the PID routine for the positioner. These factors are slightly different for different positioners. All boards are supplied with the standard set of PID settings in the controller's flash memory.

See Also

[set_pid_settings](#)

Parameters

	<i>id</i>	An identifier of a device
out	<i>pid_settings</i>	PID settings

7.1.4.78 **result_t XIMC_API** get_position (**device_t** id, **get_position_t** * the_get_position)

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

Parameters

	<i>id</i>	An identifier of a device
out	<i>the_get_position</i>	structure contains motor position.

7.1.4.79 **result_t XIMC_API** get_position_calb (**device_t** id, **get_position_calb_t** * the_get_position_calb, const **calibration_t** * calibration)

Reads position value in user units for stepper motor and encoder steps for all engines.

Parameters

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

Note

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

7.1.4.80 **result_t XIMC_API** get_power_settings (**device_t** id, **power_settings_t** * power_settings)

Read settings of step motor power control.

Used with a stepper motor only.

Parameters

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

7.1.4.81 **result_t XIMC_API** get_secure_settings (**device_t** id, **secure_settings_t** * secure_settings)

Read protection settings.

Parameters

	<i>id</i>	An identifier of a device
out	<i>secure_settings</i>	critical parameter settings to protect the hardware

See Also

status_t::flags

7.1.4.82 **result_t XIMC_API** get_serial_number (**device_t** id, unsigned int * SerialNumber)

Read device serial number.

Parameters

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

7.1.4.83 **result_t XIMC_API** get_stage_information (**device_t** id, **stage_information_t** * stage_information)

Deprecated.

Read stage information from the EEPROM.

Parameters

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

7.1.4.84 **result_t XIMC_API** get_stage_name (**device_t** id, **stage_name_t** * stage_name)

Read the user's stage name from the EEPROM.

Parameters

	<i>id</i>	An identifier of a device
out	<i>stage_name</i>	structure contains the previously set user's stage name.

7.1.4.85 **result_t XIMC_API** get_stage_settings (**device_t** id, **stage_settings_t** * stage_settings)

Deprecated.

Read stage settings from the EEPROM.

Parameters

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

7.1.4.86 **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](#)

7.1.4.87 **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](#)

7.1.4.88 **result_t XIMC_API** get_sync_in_settings (**device_t** id, **sync_in_settings_t** * sync_in_settings)

Read input synchronization settings.

This function reads the structure with a set of input synchronization settings, modes, periods and flags that specify the behavior of input synchronization. All boards are supplied with the standard set of these settings.

See Also

[set_sync_in_settings](#)

Parameters

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

7.1.4.89 **result_t XIMC_API** get_sync_in_settings_calb (**device_t** id, **sync_in_settings_calb_t** * sync_in_settings_calb, const **calibration_t** * calibration)

Read input user unit synchronization settings.

This function reads the structure with a set of input synchronization settings, modes, periods and flags that specify the behavior of input synchronization. All boards are supplied with the standard set of these settings.

See Also

[set_sync_in_settings_calb](#)

Parameters

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

7.1.4.90 **result_t XIMC_API** get_sync_out_settings (**device_t** id, **sync_out_settings_t** * sync_out_settings)

Read output synchronization settings.

This function reads the structure containing a set of output synchronization settings, modes, periods and flags that specify the behavior of output synchronization. All boards are supplied with the standard set of these settings.

See Also

[set_sync_out_settings](#)

Parameters

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

7.1.4.91 **result_t XIMC_API** get_sync_out_settings_calb (**device_t** id, **sync_out_settings_calb_t** * sync_out_settings_calb, const **calibration_t** * calibration)

Read output user unit synchronization settings.

This function reads the structure containing a set of output synchronization settings, modes, periods and flags that specify the behavior of output synchronization. All boards are supplied with the standard set of these settings.

See Also

[set_sync_in_settings_calb](#)

Parameters

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

7.1.4.92 **result_t XIMC_API** get_uart_settings (**device_t** id, **uart_settings_t** * uart_settings)

Read UART settings.

This function reads the structure containing UART settings.

See Also

[uart_settings_t](#)

Parameters

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

7.1.4.93 **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.

7.1.4.94 **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

7.1.4.95 **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](#)

7.1.4.96 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

7.1.4.97 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

7.1.4.98 void **XIMC_API** `msec_sleep` (unsigned int `msec`)

Sleeps for a specified amount of time.

Parameters

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

7.1.4.99 **device_t** **XIMC_API** `open_device` (const char * `uri`)

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

Parameters

in	<i>uri</i>	- a device URI. Device URI has a form of "xi-com:port" or "xi-net://host/serial" or "xi-emu:///abs_path_to_file". For POSIX systems one can omit root-slash in <i>abs_path_to_file</i> ; for example, "xi-emu:///home/user/virt_controller.bin". In case of USB-COM port, the "port" is the OS device URI. For example, "xi-com:\\\\.\\COM3" in Windows (note that double-backslash will be transformed to single-backslash) or "xi-com:///dev/ttyACM0" in Linux/Mac. In case of network device, the "host" is an IPv4 address or fully qualified domain 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". In case of UDP protocol, use "xi-udp://<ip/host>:<port>". For example, "xi-udp://192.168.0.1:1818". In case of virtual device, the "abs_file.to_file" is the full path to the virtual device's file. If it doesn't exist, then it is created and initialized with default values. For example, "xi-emu:///C:/dir/file.bin" in Windows or "xi-emu:///home/user/file.bin" in Linux/-Mac.
-----------	------------	---

7.1.4.100 **result_t XIMC_API** probe_device (const char * uri)

Check if a device with OS uri *uri* is XIMC device.

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

Parameters

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

7.1.4.101 **result_t XIMC_API** service_command_updf (**device_t** id)

The command switches the controller to update the firmware state.

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

7.1.4.102 **result_t XIMC_API** set_accessories_settings (**device_t** id, const **accessories_settings_t** * accessories_settings)

Deprecated.

Set additional accessories' information to the EEPROM. Can be used by the manufacturer only.

Parameters

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

7.1.4.103 **result_t XIMC_API** set_bindy_key (const char * keyfilepath)

Deprecated.

Left for compatibility Do just nothing.

7.1.4.104 **result_t XIMC_API** set_brake_settings (**device_t** id, const **brake_settings_t** * brake_settings)

Set brake control settings.

Parameters

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

7.1.4.105 **result_t XIMC_API** set_calibration_settings (**device_t** id, const **calibration_settings_t** * calibration_settings)

Set calibration settings.

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

See Also

[calibration_settings_t](#)

Parameters

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

7.1.4.106 **result_t XIMC_API** set_control_settings (**device_t** id, const **control_settings_t** * control_settings)

Read motor control settings.

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

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

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>control_settings</i>	structure contains motor control settings.

7.1.4.107 **result_t XIMC_API** set_control_settings_calb (**device_t** id, const **control_settings_calb_t** * control_settings_calb, const **calibration_t** * calibration)

Set motor control settings.

In case of CTL_MODE=1, joystick motor control is enabled. In this mode, while the joystick is maximally

displaced, the engine tends to move at `MaxSpeed[i]`. `i=0` if another value hasn't been set at the previous usage. To change the speed index "i", use the buttons.

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

Parameters

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

7.1.4.108 **result_t XIMC_API** `set_controller_name (device_t id, const controller_name_t * controller_name)`

Write user's controller name and internal settings to the FRAM.

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>controller_name</i>	structure contains the previously set user's controller name

7.1.4.109 **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 either a full path or a relative path. If the file name is set to NULL, the correction table will be cleared. File format: two tab-separated columns. Column headers are strings. Data is real, the dot is a delimiter. The first column is a coordinate. The second one is the deviation caused by a mechanical error. The maximum length of a table is 100 rows. Coordinate column must be sorted in ascending order.

See Also

[command_move](#)
[get_position_calb](#)
[get_position_calb_t](#)
[get_status_calb](#)
[status_calb_t](#)
[get_edges_settings_calb](#)
[set_edges_settings_calb](#)
[edges_settings_calb_t](#)

7.1.4.110 **result_t XIMC_API** set_ctp_settings (**device_t** id, const **ctp_settings_t** * ctp_settings)

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

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

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

Parameters

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

7.1.4.111 **result_t XIMC_API** set_debug_write (**device_t** id, const **debug_write_t** * debug_write)

Write data to firmware for debug purpose.

Manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>debug_write</i>	Debug data.

7.1.4.112 **result_t XIMC_API** set_edges_settings (**device_t** id, const **edges_settings_t** * edges_settings)

Set border and limit switches settings.

See Also

[get_edges_settings](#)

Parameters

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

7.1.4.113 **result_t XIMC_API** set_edges_settings_calb (**device_t** id, const **edges_settings_calb_t** * edges_settings_calb, const **calibration_t** * calibration)

Set border and limit switches settings in user units.

See Also

[get_edges_settings_calb](#)

Parameters

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

Note

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

7.1.4.114 **result_t XIMC_API** `set_emf_settings (device_t id, const emf_settings_t * emf_settings)`

Set electromechanical coefficients.

The settings are different for different stepper motors. Please set new settings when you change the motor.

See Also

[get_emf_settings](#)

Parameters

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

7.1.4.115 **result_t XIMC_API** `set_encoder_information (device_t id, const encoder_information_t * encoder_information)`

Deprecated.

Set encoder information to the EEPROM. Can be used by the manufacturer only.

Parameters

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

7.1.4.116 **result_t XIMC_API** `set_encoder_settings (device_t id, const encoder_settings_t * encoder_settings)`

Deprecated.

Set encoder settings to the EEPROM. Can be used by the manufacturer only.

Parameters

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

in	<i>encoder_- settings</i>	structure contains encoder settings
-----------	-------------------------------	-------------------------------------

7.1.4.117 **result_t XIMC_API** set_engine_advansed_setup (**device_t** id, const **engine_advansed_setup_t** * engine_advansed_setup)

Set engine advanced settings.

See Also

[get_engine_advansed_setup](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>engine_- advansed_setup</i>	EAS settings

7.1.4.118 **result_t XIMC_API** set_engine_settings (**device_t** id, const **engine_settings_t** * engine_settings)

Set engine settings.

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

See Also

[get_engine_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>engine_settings</i>	engine settings

7.1.4.119 **result_t XIMC_API** set_engine_settings_calb (**device_t** id, const **engine_settings_calb_t** * engine_settings_calb, const **calibration_t** * calibration)

Set user unit engine settings.

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

See Also

[get_engine_settings](#)

Parameters

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

7.1.4.120 **result_t XIMC_API** set_entype_settings (**device_t** id, const **entype_settings_t** * entype_settings)

Set engine type and driver type.

Parameters

	<i>id</i>	An identifier of a device
in	<i>entype_settings</i>	structure contains motor type and power driver type settings

7.1.4.121 **result_t XIMC_API** set_extended_settings (**device_t** id, const **extended_settings_t** * extended_settings)

Set extended settings.

Currently, it is not in use.

See Also

[get_extended_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>extended- settings</i>	EST settings

7.1.4.122 **result_t XIMC_API** set_extio_settings (**device_t** id, const **extio_settings_t** * extio_settings)

Set EXTIO settings.

This function sends the structure with a set of EXTIO settings to the controller's memory. By default, input events are signaled through a rising front, and output states are signaled by a high logic state.

See Also

[get_extio_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>extio_settings</i>	EXTIO settings

7.1.4.123 **result_t XIMC_API** set_feedback_settings (**device_t** id, const **feedback_settings_t** * feedback_settings)

Feedback settings.

Parameters

	<i>id</i>	An identifier of a device
in	<i>IPS</i>	number of encoder counts per shaft revolution. Range: 1..65535. The field is obsolete, it is recommended to write 0 to IPS and use the extended CountsPerTurn field. You may need to update the controller firmware to the latest version.
in	<i>FeedbackType</i>	type of feedback
in	<i>FeedbackFlags</i>	flags of feedback
in	<i>CountsPerTurn</i>	number of encoder counts per shaft revolution. Range: 1..4294967295. To use the CountsPerTurn field, write 0 in the IPS field, otherwise the value from the IPS field will be used.

7.1.4.124 **result_t XIMC_API** set_gear_information (**device_t** id, const **gear_information_t** * gear_information)

Deprecated.

Set gear information to the EEPROM. Can be used by the manufacturer only.

Parameters

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

7.1.4.125 **result_t XIMC_API** set_gear_settings (**device_t** id, const **gear_settings_t** * gear_settings)

Deprecated.

Set gear settings to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>gear_settings</i>	structure contains step gearhead settings

7.1.4.126 **result_t XIMC_API** set_hallsensor_information (**device_t** id, const **hallsensor_information_t** * hallsensor_information)

Deprecated.

Set hall sensor information to the EEPROM. Can be used by the manufacturer only.

Parameters

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

7.1.4.127 **result_t XIMC_API** set_hallsensor_settings (**device_t** id, const **hallsensor_settings_t** * hallsensor_settings)

Deprecated.

Set hall sensor settings to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
in	<i>hallsensor_settings</i>	structure contains hall sensor settings

7.1.4.128 **result_t XIMC_API** set_home_settings (**device_t** id, const **home_settings_t** * home_settings)

Set home settings.

This function sends home position structure to the controller's memory.

See Also

[home_settings_t](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>home_settings</i>	calibrating position settings

7.1.4.129 **result_t XIMC_API** set_home_settings_calb (**device_t** id, const **home_settings_calb_t** * home_settings_calb, const **calibration_t** * calibration)

Set user unit home settings.

This function sends home position structure to the controller's memory.

See Also

[home_settings_calb_t](#)

Parameters

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

7.1.4.130 **result_t XIMC_API** set_joystick_settings (**device_t** id, const **joystick_settings_t** * joystick_settings)

Set joystick position.

If joystick position falls outside DeadZone limits, a movement begins. The speed is defined by the joystick's position in the range of the DeadZone limit to the maximum deviation. Joystick positions inside DeadZone limits correspond to zero speed (a soft stop of motion), and positions beyond Low and High limits

correspond to `MaxSpeed[i]` or `-MaxSpeed[i]` (see command `SCTL`), where `i = 0` by default and can be changed with the left/right buttons (see command `SCTL`). If the next speed in the list is zero (both integer and microstep parts), the button press is ignored. The first speed in the list shouldn't be zero. See the Joystick control section on https://doc.xisupport.com/en/8smc5-usb/8SMCn-USB/Technical-specification/Additional_features/Joystick_control.html for more information.

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>joystick-settings</i>	structure contains joystick settings

7.1.4.131 `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
-------------------------	-----------------------------

7.1.4.132 `result_t XIMC_API set_motor_information (device_t id, const motor_information_t * motor_information)`

Deprecated.

Set motor information to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>motor-information</i>	structure contains motor information

7.1.4.133 `result_t XIMC_API set_motor_settings (device_t id, const motor_settings_t * motor_settings)`

Deprecated.

Set motor settings to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>motor_settings</i>	structure contains motor information

7.1.4.134 `result_t XIMC_API set_move_settings (device_t id, const move_settings_t * move_settings)`

Movement settings set command (speed, acceleration, threshold, etc.).

Parameters

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

7.1.4.135 **result_t XIMC_API** set_move_settings_calb (**device_t** id, const **move_settings_calb_t** * move_settings_calb, const **calibration_t** * calibration)

User unit movement settings set command (speed, acceleration, threshold, etc.).

Parameters

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

7.1.4.136 **result_t XIMC_API** set_network_settings (**device_t** id, const **network_settings_t** * network_settings)

Set network settings.

Manufacturer only. This function sets the desired network settings.

See Also

net_settings_t

Parameters

<i>DHCPEnabled</i>	DHCP enabled (1) or not (0)
<i>IPv4Address[4]</i>	Array[4] with an IP address
<i>SubnetMask[4]</i>	Array[4] with a subnet mask address
<i>Default-Gateway[4]</i>	Array[4] with a default gateway address

7.1.4.137 **result_t XIMC_API** set_nonvolatile_memory (**device_t** id, const **nonvolatile_memory_t** * nonvolatile_memory)

Write user data into the FRAM.

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>nonvolatile_memory</i>	user data.

7.1.4.138 **result_t XIMC_API** set_password_settings (**device_t** id, const **password_settings_t** * password_settings)

Sets the password.

Manufacturer only. This function sets the user password for the device's web-page.

See Also

`pwd_settings_t`

Parameters

<i>User-Password[20]</i>	Password for web-page
--------------------------	-----------------------

7.1.4.139 **result_t XIMC_API** `set_pid_settings (device_t id, const pid_settings_t * pid_settings)`

Set PID settings.

This function sends the structure with a set of PID factors to the controller's memory. These settings specify the behavior of the PID routine for the positioner. These factors are slightly different for different positioners. All boards are supplied with the standard set of PID settings in the controller's flash memory. Please use it for loading new PID settings when you change positioner. Please note that wrong PID settings lead to device malfunction.

See Also

[get_pid_settings](#)

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>pid_settings</i>	PID settings

7.1.4.140 **result_t XIMC_API** `set_position (device_t id, const set_position_t * the_set_position)`

Sets position in steps and microsteps for stepper motor.

Sets encoder position for all engines.

Parameters

	<i>id</i>	An identifier of a device
<i>out</i>	<i>the_set_position</i>	structure contains motor position.

7.1.4.141 **result_t XIMC_API** `set_position_calb (device_t id, const set_position_calb_t * the_set_position_calb, const calibration_t * calibration)`

Sets any position value and encoder value of all engines.

In user units.

Parameters

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

7.1.4.142 **result_t XIMC_API** set_power_settings (**device_t** id, const **power_settings_t** * power_settings)

Set settings of step motor power control.

Used with a stepper motor only.

Parameters

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

7.1.4.143 **result_t XIMC_API** set_secure_settings (**device_t** id, const **secure_settings_t** * secure_settings)

Set protection settings.

Parameters

	<i>id</i>	An identifier of a device
	<i>secure_settings</i>	structure with secure data

See Also

status_t::flags

7.1.4.144 **result_t XIMC_API** set_serial_number (**device_t** id, const **serial_number_t** * serial_number)

Write device serial number and hardware version to the controller's flash memory.

Along with the new serial number and hardware version, a "Key" is transmitted. The SN and hardware version are changed and saved when keys match. Can be used by the manufacturer only.

Parameters

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

7.1.4.145 **result_t XIMC_API** set_stage_information (**device_t** id, const **stage_information_t** * stage_information)

Deprecated.

Set stage information to the EEPROM. Can be used by the manufacturer only.

Parameters

	<i>id</i>	An identifier of a device
<i>in</i>	<i>stage-information</i>	structure contains stage information

7.1.4.146 **result_t XIMC_API** set_stage_name (**device_t** id, const **stage_name_t** * stage_name)

Write the user's stage name to EEPROM.

Parameters

	<i>id</i>	An identifier of a device
in	<i>stage_name</i>	structure contains the previously set user's stage name.

7.1.4.147 **result_t XIMC_API** set_stage_settings (**device_t** id, const **stage_settings_t** * stage_settings)

Deprecated.

Set stage settings to the EEPROM. Can be used by the manufacturer only

Parameters

	<i>id</i>	An identifier of a device
in	<i>stage_settings</i>	structure contains stage settings

7.1.4.148 **result_t XIMC_API** set_sync_in_settings (**device_t** id, const **sync_in_settings_t** * sync_in_settings)

Set input synchronization settings.

This function sends the structure with a set of input synchronization settings that specify the behavior of input synchronization to the controller's memory. All boards are supplied with the standard set of these settings.

See Also

[get_sync_in_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>sync_in_settings</i>	synchronization settings

7.1.4.149 **result_t XIMC_API** set_sync_in_settings_calb (**device_t** id, const **sync_in_settings_calb_t** * sync_in_settings_calb, const **calibration_t** * calibration)

Set input user unit synchronization settings.

This function sends the structure with a set of input synchronization settings that specify the behavior of input synchronization to the controller's memory. All boards are supplied with the standard set of these settings.

See Also

[get_sync_in_settings_calb](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>sync_in_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

7.1.4.150 **result_t XIMC_API** set_sync_out_settings (**device_t** id, const **sync_out_settings_t** * sync_out_settings)

Set output synchronization settings.

This function sends the structure with a set of output synchronization settings that specify the behavior of output synchronization to the controller's memory. All boards are supplied with the standard set of these settings.

See Also

[get_sync_out_settings](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>sync_out_settings</i>	synchronization settings

7.1.4.151 **result_t XIMC_API** set_sync_out_settings_calb (**device_t** id, const **sync_out_settings_calb_t** * sync_out_settings_calb, const **calibration_t** * calibration)

Set output user unit synchronization settings.

This function sends the structure with a set of output synchronization settings that specify the behavior of output synchronization to the controller's memory. All boards are supplied with the standard set of these settings.

See Also

[get_sync_in_settings_calb](#)

Parameters

	<i>id</i>	An identifier of a device
in	<i>sync_out_settings_calb</i>	synchronization settings
	<i>calibration</i>	user unit settings

7.1.4.152 **result_t XIMC_API** set_uart_settings (**device_t** id, const **uart_settings_t** * uart_settings)

Set UART settings.

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

See Also

[uart_settings_t](#)

Parameters

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

7.1.4.153 **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

7.1.4.154 **result_t XIMC_API** ximc_fix_usbser_sys (const char * device_uri)

(Deprecated) 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.

7.1.4.155 **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, [15](#)
A1Voltage_ADC
 analog_data_t, [15](#)
A2Voltage
 analog_data_t, [15](#)
A2Voltage_ADC
 analog_data_t, [15](#)
ACurrent
 analog_data_t, [16](#)
ACurrent_ADC
 analog_data_t, [16](#)
Accel
 move_settings_calb_t, [57](#)
 move_settings_t, [58](#)
accessories_settings_t, [12](#)
 LimitSwitchesSettings, [13](#)
 MBRatedCurrent, [13](#)
 MBRatedVoltage, [13](#)
 MBSettings, [13](#)
 MBTorque, [13](#)
 MagneticBrakeInfo, [13](#)
 TSGrad, [13](#)
 TSMMax, [13](#)
 TSMMin, [13](#)
 TSSettings, [14](#)
 TemperatureSensorInfo, [13](#)
Accuracy
 sync_out_settings_calb_t, [77](#)
 sync_out_settings_t, [78](#)
analog_data_t, [14](#)
 A1Voltage, [15](#)
 A1Voltage_ADC, [15](#)
 A2Voltage, [15](#)
 A2Voltage_ADC, [15](#)
 ACurrent, [16](#)
 ACurrent_ADC, [16](#)
 B1Voltage, [16](#)
 B1Voltage_ADC, [16](#)
 B2Voltage, [16](#)
 B2Voltage_ADC, [16](#)
 BCurrent, [16](#)
 BCurrent_ADC, [16](#)
 FullCurrent, [16](#)
 FullCurrent_ADC, [16](#)
 H5, [16](#)
 Joy, [16](#)
 Joy_ADC, [17](#)
 L, [17](#)
 L5, [17](#)
 L5_ADC, [17](#)
 Pot, [17](#)
 R, [17](#)
 SupVoltage, [17](#)
 SupVoltage_ADC, [17](#)
 Temp, [17](#)
 Temp_ADC, [17](#)
Antiplay
 engine_settings_calb_t, [35](#)
 engine_settings_t, [37](#)
AntiplaySpeed
 move_settings_calb_t, [57](#)
 move_settings_t, [58](#)
AveragedPowerRatio
 chart_data_t, [21](#)

B1Voltage
 analog_data_t, [16](#)
B1Voltage_ADC
 analog_data_t, [16](#)
B2Voltage
 analog_data_t, [16](#)
B2Voltage_ADC
 analog_data_t, [16](#)
BACK_EMF_KM_AUTO
 ximc.h, [105](#)
BCurrent
 analog_data_t, [16](#)
BCurrent_ADC
 analog_data_t, [16](#)
BORDER.IS.ENCODER
 ximc.h, [105](#)
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, [31](#)
BorderFlags
 edges_settings_calb_t, [29](#)
 edges_settings_t, [30](#)

brake_settings_t, 18
 BrakeFlags, 18
 t1, 18
 t2, 18
 t3, 18
 t4, 18
BrakeFlags
 brake_settings_t, 18

CONTROL_MODE_BITS
 ximc.h, 105
CONTROL_MODE_JOY
 ximc.h, 105
CONTROL_MODE_LR
 ximc.h, 106
CONTROL_MODE_OFF
 ximc.h, 106
CSS1_A
 calibration_settings_t, 19
CSS1_B
 calibration_settings_t, 19
CSS2_A
 calibration_settings_t, 19
CSS2_B
 calibration_settings_t, 19
CTP_ALARM_ON_ERROR
 ximc.h, 106
CTP_BASE
 ximc.h, 106
CTP_ENABLED
 ximc.h, 106
CTP_ERROR_CORRECTION
 ximc.h, 106
CTPFlags
 ctp_settings_t, 26
CTPMinError
 ctp_settings_t, 26
calibration_settings_t, 19
 CSS1_A, 19
 CSS1_B, 19
 CSS2_A, 19
 CSS2_B, 19
 FullCurrent_A, 19
 FullCurrent_B, 20
calibration_t, 20
chart_data_t, 20
 AveragedPowerRatio, 21
 Joy, 21
 Pot, 21
 WindingCurrentA, 21
 WindingCurrentB, 21
 WindingCurrentC, 21
 WindingVoltageA, 21
 WindingVoltageB, 22
 WindingVoltageC, 22
close_device
 ximc.h, 121
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, 121
command_eeread_settings
 ximc.h, 121
command_eesave_settings
 ximc.h, 121
command_home
 ximc.h, 121
command_homezero
 ximc.h, 122
command_left
 ximc.h, 122
command_loft
 ximc.h, 122
command_move
 ximc.h, 123
command_move_calb
 ximc.h, 123
command_movr
 ximc.h, 123
command_movr_calb
 ximc.h, 123
command_power_off
 ximc.h, 124
command_read_robust_settings
 ximc.h, 124
command_read_settings
 ximc.h, 124
command_reset
 ximc.h, 125
command_right
 ximc.h, 125
command_save_robust_settings
 ximc.h, 125
command_save_settings
 ximc.h, 125
command_sstp
 ximc.h, 125
command_start_measurements
 ximc.h, 125
command_stop
 ximc.h, 126
command_update_firmware
 ximc.h, 126
command_wait_for_stop
 ximc.h, 126
command_zero
 ximc.h, 126
control_settings_calb_t, 22

- Flags, [23](#)
- MaxClickTime, [23](#)
- MaxSpeed, [23](#)
- Timeout, [23](#)
- control_settings_t, [23](#)
 - Flags, [24](#)
 - MaxClickTime, [24](#)
 - MaxSpeed, [24](#)
 - Timeout, [24](#)
 - uDeltaPosition, [24](#)
 - uMaxSpeed, [24](#)
- controller_name_t, [25](#)
 - ControllerName, [25](#)
 - CtrlFlags, [25](#)
- ControllerName
 - controller_name_t, [25](#)
- CountsPerTurn
 - feedback_settings_t, [40](#)
- 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, [25](#)
 - CTPFlags, [26](#)
 - CTPMinError, [26](#)
- CtrlFlags
 - controller_name_t, [25](#)
- 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](#)
- DHCPEnabled
 - network_settings_t, [59](#)
- DRIVER_TYPE_EXTERNAL
 - ximc.h, [106](#)
- DeadZone
 - joystick_settings_t, [50](#)
- debug_read_t, [26](#)
 - DebugData, [26](#)
- debug_write_t, [27](#)
 - DebugData, [27](#)
- DebugData
 - debug_read_t, [26](#)
 - debug_write_t, [27](#)
- Decel
 - move_settings_calb_t, [57](#)
 - move_settings_t, [58](#)
- DefaultGateway
 - network_settings_t, [59](#)
- DetentTorque
 - motor_settings_t, [54](#)
- device_information_t, [27](#)
 - Major, [28](#)
 - Minor, [28](#)
 - Release, [28](#)
- device_network_information_t, [28](#)
- DriverType
 - entype_settings_t, [38](#)
- EEPROM_PRECEDENCE
 - ximc.h, [106](#)
- ENC_STATE_ABSENT
 - ximc.h, [106](#)
- ENC_STATE_MALFUNC
 - ximc.h, [107](#)
- ENC_STATE_OK
 - ximc.h, [107](#)
- ENC_STATE_REVERS
 - ximc.h, [107](#)
- 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, [108](#)
- ENGINE_LIMIT_RPM
 - ximc.h, [108](#)
- ENGINE_LIMIT_VOLT
 - ximc.h, [108](#)
- 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, [109](#)
- EXTIO_SETUP_INVERT
 - ximc.h, [109](#)
- EXTIO_SETUP_OUTPUT
 - ximc.h, [110](#)
- EXTIOModeFlags
 - extio_settings_t, [39](#)
- EXTIOSetupFlags
 - extio_settings_t, [39](#)
- edges_settings_calb_t, [28](#)
 - BorderFlags, [29](#)
 - EnderFlags, [29](#)
 - LeftBorder, [29](#)
 - RightBorder, [29](#)
- edges_settings_t, [29](#)
 - BorderFlags, [30](#)
 - EnderFlags, [30](#)
 - LeftBorder, [30](#)
 - RightBorder, [30](#)
 - uLeftBorder, [30](#)
 - uRightBorder, [30](#)
- Efficiency
 - gear_settings_t, [42](#)
- emf_settings_t, [31](#)
 - BackEMFFlags, [31](#)
 - Km, [31](#)
 - L, [31](#)
 - R, [31](#)
- EncPosition
 - get_position_calb_t, [43](#)
 - get_position_t, [44](#)
 - 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, [32](#)
 - Manufacturer, [32](#)
 - PartNumber, [32](#)
- encoder_settings_t, [32](#)
 - EncoderSettings, [33](#)
 - MaxCurrentConsumption, [33](#)
 - MaxOperatingFrequency, [33](#)
 - SupplyVoltageMax, [33](#)
 - SupplyVoltageMin, [33](#)
- EncoderSettings
 - encoder_settings_t, [33](#)
- EnderFlags
 - edges_settings_calb_t, [29](#)
 - edges_settings_t, [30](#)
- engine_advanced_setup_t, [34](#)
 - stepcloseloop_Kp_high, [34](#)
 - stepcloseloop_Kp_low, [34](#)
 - stepcloseloop_Kw, [34](#)
- engine_settings_calb_t, [34](#)
 - Antiplay, [35](#)
 - EngineFlags, [35](#)
 - MicrostepMode, [35](#)
 - NomCurrent, [35](#)
 - NomSpeed, [35](#)
 - NomVoltage, [36](#)
 - StepsPerRev, [36](#)
- engine_settings_t, [36](#)
 - Antiplay, [37](#)
 - EngineFlags, [37](#)
 - MicrostepMode, [37](#)
 - NomCurrent, [37](#)
 - NomSpeed, [37](#)
 - NomVoltage, [37](#)
 - StepsPerRev, [37](#)
 - uNomSpeed, [37](#)
- EngineFlags
 - engine_settings_calb_t, [35](#)
 - engine_settings_t, [37](#)
- EngineType
 - entype_settings_t, [38](#)
- entype_settings_t, [38](#)
 - DriverType, [38](#)
 - EngineType, [38](#)
- enumerate_devices
 - ximc.h, [127](#)
- Error
 - measurements_t, [51](#)
- ExpFactor
 - joystick_settings_t, [50](#)
- extended_settings_t, [38](#)
- extio_settings_t, [39](#)
 - EXTIOModeFlags, [39](#)
 - EXTIOSetupFlags, [39](#)
- FEEDBACK_EMF
 - ximc.h, [110](#)
- FEEDBACK_ENC_REVERSE
 - ximc.h, [110](#)
- FEEDBACK_ENCODER
 - ximc.h, [110](#)
- FEEDBACK_NONE
 - ximc.h, [111](#)
- FastHome
 - home_settings_calb_t, [47](#)
 - home_settings_t, [48](#)
- feedback_settings_t, [39](#)
 - CountsPerTurn, [40](#)
 - FeedbackFlags, [40](#)

- FeedbackType, [40](#)
- IPS, [40](#)
- FeedbackFlags
 - feedback_settings_t, [40](#)
- FeedbackType
 - feedback_settings_t, [40](#)
- Flags
 - control_settings_calb_t, [23](#)
 - control_settings_t, [24](#)
 - secure_settings_t, [64](#)
 - status_calb_t, [71](#)
 - status_t, [73](#)
- free_enumerate_devices
 - ximc.h, [127](#)
- FullCurrent
 - analog_data_t, [16](#)
- FullCurrent_A
 - calibration_settings_t, [19](#)
- FullCurrent_ADC
 - analog_data_t, [16](#)
- FullCurrent_B
 - calibration_settings_t, [20](#)
- GPIOFlags
 - status_calb_t, [71](#)
 - status_t, [73](#)
- gear_information_t, [40](#)
 - Manufacturer, [41](#)
 - PartNumber, [41](#)
- gear_settings_t, [41](#)
 - Efficiency, [42](#)
 - InputInertia, [42](#)
 - MaxOutputBacklash, [42](#)
 - RatedInputSpeed, [42](#)
 - RatedInputTorque, [42](#)
 - ReductionIn, [42](#)
 - ReductionOut, [42](#)
- get_accessories_settings
 - ximc.h, [127](#)
- get_analog_data
 - ximc.h, [128](#)
- get_bootloader_version
 - ximc.h, [128](#)
- get_brake_settings
 - ximc.h, [128](#)
- get_calibration_settings
 - ximc.h, [128](#)
- get_chart_data
 - ximc.h, [128](#)
- get_control_settings
 - ximc.h, [129](#)
- get_control_settings_calb
 - ximc.h, [129](#)
- get_controller_name
 - ximc.h, [129](#)
- get_ctp_settings
 - ximc.h, [130](#)
- get_debug_read
 - ximc.h, [130](#)
- get_device_count
 - ximc.h, [130](#)
- get_device_information
 - ximc.h, [130](#)
- get_device_name
 - ximc.h, [131](#)
- get_edges_settings
 - ximc.h, [131](#)
- get_edges_settings_calb
 - ximc.h, [131](#)
- get_emf_settings
 - ximc.h, [132](#)
- get_encoder_information
 - ximc.h, [132](#)
- get_encoder_settings
 - ximc.h, [132](#)
- get_engine_advanced_setup
 - ximc.h, [132](#)
- get_engine_settings
 - ximc.h, [133](#)
- get_engine_settings_calb
 - ximc.h, [133](#)
- get_entype_settings
 - ximc.h, [133](#)
- get_enumerate_device_controller_name
 - ximc.h, [133](#)
- get_enumerate_device_information
 - ximc.h, [134](#)
- get_enumerate_device_network_information
 - ximc.h, [134](#)
- get_enumerate_device_serial
 - ximc.h, [134](#)
- get_enumerate_device_stage_name
 - ximc.h, [135](#)
- get_extended_settings
 - ximc.h, [135](#)
- get_extio_settings
 - ximc.h, [135](#)
- get_feedback_settings
 - ximc.h, [135](#)
- get_firmware_version
 - ximc.h, [136](#)
- get_gear_information
 - ximc.h, [136](#)
- get_gear_settings
 - ximc.h, [136](#)
- get_globally_unique_identifier
 - ximc.h, [136](#)
- get_hallsensor_information
 - ximc.h, [137](#)
- get_hallsensor_settings
 - ximc.h, [137](#)
- get_home_settings

- ximc.h, [137](#)
- get_home_settings_calb
 - ximc.h, [137](#)
- get_init_random
 - ximc.h, [138](#)
- get_joystick_settings
 - ximc.h, [138](#)
- get_measurements
 - ximc.h, [138](#)
- get_motor_information
 - ximc.h, [139](#)
- get_motor_settings
 - ximc.h, [139](#)
- get_move_settings
 - ximc.h, [139](#)
- get_move_settings_calb
 - ximc.h, [139](#)
- get_network_settings
 - ximc.h, [140](#)
- get_nonvolatile_memory
 - ximc.h, [140](#)
- get_password_settings
 - ximc.h, [140](#)
- get_pid_settings
 - ximc.h, [140](#)
- get_position
 - ximc.h, [141](#)
- get_position_calb
 - ximc.h, [141](#)
- get_position_calb_t, [42](#)
 - EncPosition, [43](#)
 - Position, [43](#)
- get_position_t, [43](#)
 - EncPosition, [44](#)
 - uPosition, [44](#)
- get_power_settings
 - ximc.h, [141](#)
- get_secure_settings
 - ximc.h, [141](#)
- get_serial_number
 - ximc.h, [142](#)
- get_stage_information
 - ximc.h, [142](#)
- get_stage_name
 - ximc.h, [142](#)
- get_stage_settings
 - ximc.h, [142](#)
- get_status
 - ximc.h, [143](#)
- get_status_calb
 - ximc.h, [143](#)
- get_sync_in_settings
 - ximc.h, [143](#)
- get_sync_in_settings_calb
 - ximc.h, [143](#)
- get_sync_out_settings
 - ximc.h, [144](#)
- get_sync_out_settings_calb
 - ximc.h, [144](#)
- get_uart_settings
 - ximc.h, [145](#)
- globally_unique_identifier_t, [44](#)
 - UniquelD0, [44](#)
 - UniquelD1, [44](#)
 - UniquelD2, [44](#)
 - UniquelD3, [44](#)
- goto_firmware
 - ximc.h, [145](#)
- H5
 - analog_data_t, [16](#)
- H_BRIDGE_ALERT
 - ximc.h, [111](#)
- 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, [45](#)
 - Manufacturer, [45](#)
 - PartNumber, [45](#)
- hallsensor_settings_t, [45](#)
 - MaxCurrentConsumption, [46](#)
 - MaxOperatingFrequency, [46](#)
 - SupplyVoltageMax, [46](#)
 - SupplyVoltageMin, [46](#)
- has_firmware
 - ximc.h, [145](#)
- HoldCurrent
 - power_settings_t, [62](#)
- home_settings_calb_t, [46](#)
 - FastHome, [47](#)
 - HomeDelta, [47](#)
 - HomeFlags, [47](#)
 - SlowHome, [47](#)
- home_settings_t, [47](#)
 - FastHome, [48](#)
 - HomeDelta, [48](#)
 - HomeFlags, [48](#)
 - SlowHome, [48](#)
 - uFastHome, [48](#)

- uHomeDelta, [49](#)
- uSlowHome, [49](#)
- HomeDelta
 - home_settings_calb_t, [47](#)
 - home_settings_t, [48](#)
- HomeFlags
 - home_settings_calb_t, [47](#)
 - home_settings_t, [48](#)
- HorizontalLoadCapacity
 - stage_settings_t, [68](#)
- IPS
 - feedback_settings_t, [40](#)
- IPv4Address
 - network_settings_t, [59](#)
- init_random_t, [49](#)
 - key, [49](#)
- InputInertia
 - gear_settings_t, [42](#)
- 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, [16](#)
 - chart_data_t, [21](#)
- Joy_ADC
 - analog_data_t, [17](#)
- JoyCenter
 - joystick_settings_t, [50](#)
- JoyFlags
 - joystick_settings_t, [50](#)
- JoyHighEnd
 - joystick_settings_t, [51](#)
- JoyLowEnd
 - joystick_settings_t, [51](#)
- joystick_settings_t, [49](#)
 - DeadZone, [50](#)
 - ExpFactor, [50](#)
 - JoyCenter, [50](#)
 - JoyFlags, [50](#)
 - JoyHighEnd, [51](#)
 - JoyLowEnd, [51](#)
- Key
 - serial_number_t, [64](#)
- key
 - init_random_t, [49](#)
- Km
 - emf_settings_t, [31](#)
- L
 - analog_data_t, [17](#)
 - emf_settings_t, [31](#)
- L5
 - analog_data_t, [17](#)
- L5_ADC
 - analog_data_t, [17](#)
- LOW_UPWR_PROTECTION
 - ximc.h, [112](#)
- LeadScrewPitch
 - stage_settings_t, [68](#)
- LeftBorder
 - edges_settings_calb_t, [29](#)
 - edges_settings_t, [30](#)
- Length
 - measurements_t, [51](#)
- LimitSwitchesSettings
 - accessories_settings_t, [13](#)
- load_correction_table
 - ximc.h, [145](#)
- logging_callback_stderr_narrow
 - ximc.h, [146](#)
- logging_callback_stderr_wide
 - ximc.h, [146](#)
- logging_callback_t
 - ximc.h, [120](#)
- LowUpwrOff
 - secure_settings_t, [64](#)
- MBRatedCurrent
 - accessories_settings_t, [13](#)
- MBRatedVoltage
 - accessories_settings_t, [13](#)
- MBSettings
 - accessories_settings_t, [13](#)
- MBTorque
 - accessories_settings_t, [13](#)
- 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, [114](#)
- MVCMD_MOVR
 - ximc.h, [114](#)
- MVCMD_NAME_BITS
 - ximc.h, [114](#)

- 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, [13](#)
- Major
 - device_information_t, [28](#)
 - serial_number_t, [64](#)
- Manufacturer
 - encoder_information_t, [32](#)
 - gear_information_t, [41](#)
 - hallsensor_information_t, [45](#)
 - motor_information_t, [52](#)
 - stage_information_t, [67](#)
- MaxClickTime
 - control_settings_calb_t, [23](#)
 - control_settings_t, [24](#)
- MaxCurrent
 - motor_settings_t, [54](#)
- MaxCurrentConsumption
 - encoder_settings_t, [33](#)
 - hallsensor_settings_t, [46](#)
 - stage_settings_t, [69](#)
- MaxCurrentTime
 - motor_settings_t, [54](#)
- MaxOperatingFrequency
 - encoder_settings_t, [33](#)
 - hallsensor_settings_t, [46](#)
- MaxOutputBacklash
 - gear_settings_t, [42](#)
- MaxSpeed
 - control_settings_calb_t, [23](#)
 - control_settings_t, [24](#)
 - motor_settings_t, [54](#)
 - stage_settings_t, [69](#)
- measurements_t, [51](#)
 - Error, [51](#)
 - Length, [51](#)
 - Speed, [51](#)
- MechanicalTimeConstant
 - motor_settings_t, [54](#)
- MicrostepMode
 - engine_settings_calb_t, [35](#)
 - engine_settings_t, [37](#)
- MinimumUusb
 - secure_settings_t, [64](#)
- Minor
 - device_information_t, [28](#)
 - serial_number_t, [65](#)
- motor_information_t, [52](#)
 - Manufacturer, [52](#)
 - PartNumber, [52](#)
- motor_settings_t, [52](#)
 - DetentTorque, [54](#)
 - MaxCurrent, [54](#)
 - MaxCurrentTime, [54](#)
 - MaxSpeed, [54](#)
 - MechanicalTimeConstant, [54](#)
 - MotorType, [54](#)
 - NoLoadCurrent, [54](#)
 - NoLoadSpeed, [54](#)
 - NominalCurrent, [55](#)
 - NominalPower, [55](#)
 - NominalSpeed, [55](#)
 - NominalTorque, [55](#)
 - NominalVoltage, [55](#)
 - Phases, [55](#)
 - Poles, [55](#)
 - RotorInertia, [55](#)
 - SpeedConstant, [55](#)
 - SpeedTorqueGradient, [56](#)
 - StallTorque, [56](#)
 - TorqueConstant, [56](#)
 - WindingInductance, [56](#)
 - WindingResistance, [56](#)
- MotorType
 - motor_settings_t, [54](#)
- move_settings_calb_t, [56](#)
 - Accel, [57](#)
 - AntiplaySpeed, [57](#)
 - Decel, [57](#)
 - MoveFlags, [57](#)
 - Speed, [57](#)
- move_settings_t, [57](#)
 - Accel, [58](#)
 - AntiplaySpeed, [58](#)
 - Decel, [58](#)
 - MoveFlags, [58](#)
 - Speed, [58](#)
 - uAntiplaySpeed, [58](#)
 - uSpeed, [59](#)
- MoveFlags
 - move_settings_calb_t, [57](#)
 - move_settings_t, [58](#)
- MoveSts
 - status_calb_t, [71](#)
 - status_t, [74](#)
- msec_sleep
 - ximc.h, [146](#)
- MvCmdSts
 - status_calb_t, [71](#)
 - status_t, [74](#)
- network_settings_t, [59](#)
 - DHCPEnabled, [59](#)

- DefaultGateway, [59](#)
- IPv4Address, [59](#)
- SubnetMask, [60](#)
- NoLoadCurrent
 - motor_settings_t, [54](#)
- NoLoadSpeed
 - motor_settings_t, [54](#)
- NomCurrent
 - engine_settings_calb_t, [35](#)
 - engine_settings_t, [37](#)
- NomSpeed
 - engine_settings_calb_t, [35](#)
 - engine_settings_t, [37](#)
- NomVoltage
 - engine_settings_calb_t, [36](#)
 - engine_settings_t, [37](#)
- NominalCurrent
 - motor_settings_t, [55](#)
- NominalPower
 - motor_settings_t, [55](#)
- NominalSpeed
 - motor_settings_t, [55](#)
- NominalTorque
 - motor_settings_t, [55](#)
- NominalVoltage
 - motor_settings_t, [55](#)
- nonvolatile_memory_t, [60](#)
 - UserData, [60](#)
- open_device
 - ximc.h, [146](#)
- POWER_OFF_ENABLED
 - ximc.h, [114](#)
- POWER_REDUCE_ENABLED
 - ximc.h, [114](#)
- POWER_SMOOTH_CURRENT
 - ximc.h, [114](#)
- PWR_STATE_MAX
 - ximc.h, [114](#)
- PWR_STATE_NORM
 - ximc.h, [115](#)
- PWR_STATE_OFF
 - ximc.h, [115](#)
- PWR_STATE_REDUCE
 - ximc.h, [115](#)
- PWR_STATE_UNKNOWN
 - ximc.h, [115](#)
- PWRSts
 - status_calb_t, [71](#)
 - status_t, [74](#)
- PartNumber
 - encoder_information_t, [32](#)
 - gear_information_t, [41](#)
 - hallsensor_information_t, [45](#)
 - motor_information_t, [52](#)
 - stage_information_t, [67](#)
- password_settings_t, [60](#)
 - UserPassword, [61](#)
- Phases
 - motor_settings_t, [55](#)
- pid_settings_t, [61](#)
- Poles
 - motor_settings_t, [55](#)
- PosFlags
 - set_position_calb_t, [65](#)
 - set_position_t, [66](#)
- Position
 - get_position_calb_t, [43](#)
 - set_position_calb_t, [65](#)
 - sync_in_settings_calb_t, [75](#)
- PositionerName
 - stage_name_t, [68](#)
- Pot
 - analog_data_t, [17](#)
 - chart_data_t, [21](#)
- 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, [147](#)
- R
 - analog_data_t, [17](#)
 - emf_settings_t, [31](#)
- REV_SENS_INV
 - ximc.h, [115](#)
- RPM_DIV_1000
 - ximc.h, [115](#)
- RatedInputSpeed
 - gear_settings_t, [42](#)
- RatedInputTorque
 - gear_settings_t, [42](#)
- ReductionIn
 - gear_settings_t, [42](#)
- ReductionOut
 - gear_settings_t, [42](#)
- Release
 - device_information_t, [28](#)
 - serial_number_t, [65](#)
- RightBorder
 - edges_settings_calb_t, [29](#)
 - edges_settings_t, [30](#)
- RotorInertia
 - motor_settings_t, [55](#)
- SN

serial_number_t, [65](#)
STATE_ALARM
 ximc.h, [115](#)
STATE_BRAKE
 ximc.h, [115](#)
STATE_BUTTON_LEFT
 ximc.h, [116](#)
STATE_BUTTON_RIGHT
 ximc.h, [116](#)
STATE_CONTR
 ximc.h, [116](#)
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, [117](#)
STATE_ERRD
 ximc.h, [117](#)
STATE_ERRV
 ximc.h, [117](#)
STATE_EXTIO_ALARM
 ximc.h, [117](#)
STATE_GPIO_LEVEL
 ximc.h, [117](#)
STATE_GPIO_PINOUT
 ximc.h, [117](#)
STATE_IS_HOMED
 ximc.h, [117](#)
STATE_LEFT_EDGE
 ximc.h, [117](#)
STATE_POWER_OVERHEAT
 ximc.h, [118](#)
STATE_REV_SENSOR
 ximc.h, [118](#)
STATE_RIGHT_EDGE
 ximc.h, [118](#)
STATE_SECUR
 ximc.h, [118](#)
STATE_SYNC_INPUT
 ximc.h, [118](#)
STATE_SYNC_OUTPUT
 ximc.h, [118](#)
SYNCIN_ENABLED
 ximc.h, [119](#)
SYNCIN_GOTOPOSITION
 ximc.h, [119](#)
SYNCIN_INVERT
 ximc.h, [119](#)
SYNCOUT_ENABLED
 ximc.h, [119](#)
SYNCOUT_IN_STEPS
 ximc.h, [119](#)
SYNCOUT_INVERT
 ximc.h, [119](#)
SYNCOUT_ONPERIOD
 ximc.h, [119](#)
SYNCOUT_ONSTART
 ximc.h, [119](#)
SYNCOUT_ONSTOP
 ximc.h, [119](#)
SYNCOUT_STATE
 ximc.h, [119](#)
secure_settings_t, [62](#)
 CriticalIpwr, [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, [147](#)
set_accessories_settings
 ximc.h, [147](#)
set_bindy_key
 ximc.h, [147](#)
set_brake_settings
 ximc.h, [147](#)
set_calibration_settings
 ximc.h, [148](#)
set_control_settings
 ximc.h, [148](#)
set_control_settings_calb
 ximc.h, [148](#)
set_controller_name
 ximc.h, [149](#)
set_correction_table
 ximc.h, [149](#)
set_ctp_settings
 ximc.h, [149](#)
set_debug_write
 ximc.h, [150](#)
set_edges_settings
 ximc.h, [150](#)
set_edges_settings_calb
 ximc.h, [150](#)
set_emf_settings
 ximc.h, [151](#)
set_encoder_information
 ximc.h, [151](#)

- set_encoder_settings
 - ximc.h, [151](#)
- set_engine_advanced_setup
 - ximc.h, [152](#)
- set_engine_settings
 - ximc.h, [152](#)
- set_engine_settings_calb
 - ximc.h, [152](#)
- set_entype_settings
 - ximc.h, [153](#)
- set_extended_settings
 - ximc.h, [153](#)
- set_extio_settings
 - ximc.h, [153](#)
- set_feedback_settings
 - ximc.h, [153](#)
- set_gear_information
 - ximc.h, [154](#)
- set_gear_settings
 - ximc.h, [154](#)
- set_hallsensor_information
 - ximc.h, [154](#)
- set_hallsensor_settings
 - ximc.h, [154](#)
- set_home_settings
 - ximc.h, [155](#)
- set_home_settings_calb
 - ximc.h, [155](#)
- set_joystick_settings
 - ximc.h, [155](#)
- set_logging_callback
 - ximc.h, [156](#)
- set_motor_information
 - ximc.h, [156](#)
- set_motor_settings
 - ximc.h, [156](#)
- set_move_settings
 - ximc.h, [156](#)
- set_move_settings_calb
 - ximc.h, [157](#)
- set_network_settings
 - ximc.h, [157](#)
- set_nonvolatile_memory
 - ximc.h, [157](#)
- set_password_settings
 - ximc.h, [157](#)
- set_pid_settings
 - ximc.h, [158](#)
- set_position
 - ximc.h, [158](#)
- set_position_calb
 - ximc.h, [158](#)
- 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, [158](#)
- set_secure_settings
 - ximc.h, [159](#)
- set_serial_number
 - ximc.h, [159](#)
- set_stage_information
 - ximc.h, [159](#)
- set_stage_name
 - ximc.h, [159](#)
- set_stage_settings
 - ximc.h, [160](#)
- set_sync_in_settings
 - ximc.h, [160](#)
- set_sync_in_settings_calb
 - ximc.h, [160](#)
- set_sync_out_settings
 - ximc.h, [161](#)
- set_sync_out_settings_calb
 - ximc.h, [161](#)
- set_uart_settings
 - ximc.h, [161](#)
- SlowHome
 - home_settings_calb_t, [47](#)
 - home_settings_t, [48](#)
- Speed
 - measurements_t, [51](#)
 - move_settings_calb_t, [57](#)
 - move_settings_t, [58](#)
 - sync_in_settings_calb_t, [75](#)
 - sync_in_settings_t, [76](#)
- SpeedConstant
 - motor_settings_t, [55](#)
- SpeedTorqueGradient
 - motor_settings_t, [56](#)
- 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, [56](#)

- 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.advansed_setup_t, 34
- stepcloseloop_Kp_low
 - engine.advansed_setup_t, 34
- stepcloseloop_Kw
 - engine.advansed_setup_t, 34
- StepsPerRev
 - engine_settings_calb_t, 36
 - engine_settings_t, 37
- SubnetMask
 - network_settings_t, 60
- SupVoltage
 - analog_data_t, 17
- SupVoltage_ADC
 - analog_data_t, 17
- SupplyVoltageMax
 - encoder_settings_t, 33
 - hallsensor_settings_t, 46
 - stage_settings_t, 69
- SupplyVoltageMin
 - encoder_settings_t, 33
 - hallsensor_settings_t, 46
 - 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, 18
- t2
 - brake_settings_t, 18
- t3
 - brake_settings_t, 18
- t4
 - brake_settings_t, 18
- TSGrad
 - accessories_settings_t, 13
- TSMaX
 - accessories_settings_t, 13
- TSMIn
 - accessories_settings_t, 13
- TSSettings
 - accessories_settings_t, 14
- Temp
 - analog_data_t, 17

- Temp_ADC
 - analog_data.t, [17](#)
- TemperatureSensorInfo
 - accessories_settings.t, [13](#)
- Timeout
 - control_settings_calb.t, [23](#)
 - control_settings.t, [24](#)
- TorqueConstant
 - motor_settings.t, [56](#)
- 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, [58](#)
- uCurPosition
 - status.t, [74](#)
- uCurSpeed
 - status.t, [74](#)
- uDeltaPosition
 - control_settings.t, [24](#)
- uFastHome
 - home_settings.t, [48](#)
- uHomeDelta
 - home_settings.t, [49](#)
- uLeftBorder
 - edges_settings.t, [30](#)
- uMaxSpeed
 - control_settings.t, [24](#)
- uNomSpeed
 - engine_settings.t, [37](#)
- uPosition
 - get_position.t, [44](#)
 - set_position.t, [66](#)
 - sync_in_settings.t, [76](#)
- uRightBorder
 - edges_settings.t, [30](#)
- uSlowHome
 - home_settings.t, [49](#)
- uSpeed
 - move_settings.t, [59](#)
 - sync_in_settings.t, [76](#)
- uart_settings.t, [79](#)
- UARTSetupFlags, [79](#)
- UniqueID0
 - globally_unique_identifier.t, [44](#)
- UniqueID1
 - globally_unique_identifier.t, [44](#)
- UniqueID2
 - globally_unique_identifier.t, [44](#)
- UniqueID3
 - globally_unique_identifier.t, [44](#)
- Units
 - stage_settings.t, [69](#)
- Upwr
 - status_calb.t, [71](#)
 - status.t, [74](#)
- UserData
 - nonvolatile_memory.t, [60](#)
- UserPassword
 - password_settings.t, [61](#)
- Uusb
 - status_calb.t, [72](#)
 - status.t, [74](#)
- VerticalLoadCapacity
 - stage_settings.t, [69](#)
- WIND_A_STATE_ABSENT
 - ximc.h, [120](#)
- WIND_A_STATE_OK
 - ximc.h, [120](#)
- WIND_B_STATE_ABSENT
 - ximc.h, [120](#)
- WIND_B_STATE_OK
 - ximc.h, [120](#)
- WindSts
 - status_calb.t, [72](#)
 - status.t, [74](#)
- WindingCurrentA
 - chart_data.t, [21](#)
- WindingCurrentB
 - chart_data.t, [21](#)
- WindingCurrentC
 - chart_data.t, [21](#)
- WindingInductance
 - motor_settings.t, [56](#)
- WindingResistance
 - motor_settings.t, [56](#)
- WindingVoltageA
 - chart_data.t, [21](#)
- WindingVoltageB
 - chart_data.t, [22](#)
- WindingVoltageC
 - chart_data.t, [22](#)
- write_key
 - ximc.h, [162](#)
- XIMC_API
 - ximc.h, [120](#)
- ximc.h, [80](#)
 - BACK_EMF_KM_AUTO, [105](#)
 - BORDER_IS_ENCODER, [105](#)
 - 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, 106
CONTROL_MODE_OFF, 106
CTP_ALARM_ON_ERROR, 106
CTP_BASE, 106
CTP_ENABLED, 106
close_device, 121
command_clear_fram, 121
command_eeread_settings, 121
command_eesave_settings, 121
command_home, 121
command_homezero, 122
command_left, 122
command_loft, 122
command_move, 123
command_move_calb, 123
command_movr, 123
command_movr_calb, 123
command_power_off, 124
command_read_robust_settings, 124
command_read_settings, 124
command_reset, 125
command_right, 125
command_save_robust_settings, 125
command_save_settings, 125
command_sstp, 125
command_start_measurements, 125
command_stop, 126
command_update_firmware, 126
command_wait_for_stop, 126
command_zero, 126
EEPROM_PRECEDENCE, 106
ENC_STATE_ABSENT, 106
ENC_STATE_MALFUNC, 107
ENC_STATE_OK, 107
ENC_STATE_REVERS, 107
ENC_STATE_UNKNOWN, 107
ENDER_SWAP, 107
ENGINE_ACCEL_ON, 107
ENGINE_ANTIPLAY, 107
ENGINE_LIMIT_CURR, 108
ENGINE_LIMIT_RPM, 108
ENGINE_LIMIT_VOLT, 108
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, 109
EXTIO_SETUP_INVERT, 109
EXTIO_SETUP_OUTPUT, 110
enumerate_devices, 127
FEEDBACK_EMF, 110
FEEDBACK_ENCODER, 110
FEEDBACK_NONE, 111
free_enumerate_devices, 127
get_accessories_settings, 127
get_analog_data, 128
get_bootloader_version, 128
get_brake_settings, 128
get_calibration_settings, 128
get_chart_data, 128
get_control_settings, 129
get_control_settings_calb, 129
get_controller_name, 129
get_ctp_settings, 130
get_debug_read, 130
get_device_count, 130
get_device_information, 130
get_device_name, 131
get_edges_settings, 131
get_edges_settings_calb, 131
get_emf_settings, 132
get_encoder_information, 132
get_encoder_settings, 132
get_engine_advanced_setup, 132
get_engine_settings, 133
get_engine_settings_calb, 133
get_entype_settings, 133
get_enumerate_device_controller_name, 133
get_enumerate_device_information, 134
get_enumerate_device_network_information, 134
get_enumerate_device_serial, 134
get_enumerate_device_stage_name, 135
get_extended_settings, 135
get_extio_settings, 135
get_feedback_settings, 135
get_firmware_version, 136
get_gear_information, 136
get_gear_settings, 136
get_globally_unique_identifier, 136
get_hallsensor_information, 137
get_hallsensor_settings, 137
get_home_settings, 137
get_home_settings_calb, 137
get_init_random, 138
get_joystick_settings, 138
get_measurements, 138
get_motor_information, 139
get_motor_settings, 139
get_move_settings, 139
get_move_settings_calb, 139
get_network_settings, 140
get_nonvolatile_memory, 140
get_password_settings, 140
get_pid_settings, 140
get_position, 141
get_position_calb, 141
get_power_settings, 141
get_secure_settings, 141

get_serial_number, [142](#)
get_stage_information, [142](#)
get_stage_name, [142](#)
get_stage_settings, [142](#)
get_status, [143](#)
get_status_calb, [143](#)
get_sync_in_settings, [143](#)
get_sync_in_settings_calb, [143](#)
get_sync_out_settings, [144](#)
get_sync_out_settings_calb, [144](#)
get_uart_settings, [145](#)
goto_firmware, [145](#)
H_BRIDGE_ALERT, [111](#)
HOME_DIR_FIRST, [111](#)
HOME_DIR_SECOND, [111](#)
HOME_HALF_MV, [111](#)
HOME_MV_SEC_EN, [111](#)
HOME_USE_FAST, [112](#)
has_firmware, [145](#)
JOY_REVERSE, [112](#)
LOW_UPWR_PROTECTION, [112](#)
load_correction_table, [145](#)
logging_callback_stderr_narrow, [146](#)
logging_callback_stderr_wide, [146](#)
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, [114](#)
MVCMD_MOVR, [114](#)
MVCMD_NAME_BITS, [114](#)
MVCMD_RIGHT, [114](#)
MVCMD_RUNNING, [114](#)
MVCMD_SSTP, [114](#)
MVCMD_STOP, [114](#)
MVCMD_UKNWN, [114](#)
msec_sleep, [146](#)
open_device, [146](#)
POWER_OFF_ENABLED, [114](#)
PWR_STATE_MAX, [114](#)
PWR_STATE_NORM, [115](#)
PWR_STATE_OFF, [115](#)
PWR_STATE_REDUCT, [115](#)
PWR_STATE_UNKNOWN, [115](#)
probe_device, [147](#)
REV_SENS_INV, [115](#)
RPM_DIV_1000, [115](#)
STATE_ALARM, [115](#)
STATE_BRAKE, [115](#)
STATE_BUTTON_LEFT, [116](#)
STATE_BUTTON_RIGHT, [116](#)
STATE_CONTR, [116](#)
STATE_CTP_ERROR, [116](#)
STATE_DIG_SIGNAL, [116](#)
STATE_ENC_A, [116](#)
STATE_ENC_B, [116](#)
STATE_ERRC, [117](#)
STATE_ERRD, [117](#)
STATE_ERRV, [117](#)
STATE_EXTIO_ALARM, [117](#)
STATE_GPIO_LEVEL, [117](#)
STATE_GPIO_PINOUT, [117](#)
STATE_IS_HOMED, [117](#)
STATE_LEFT_EDGE, [117](#)
STATE_REV_SENSOR, [118](#)
STATE_RIGHT_EDGE, [118](#)
STATE_SECUR, [118](#)
STATE_SYNC_INPUT, [118](#)
STATE_SYNC_OUTPUT, [118](#)
SYNCIN_ENABLED, [119](#)
SYNCIN_GOTOPOSITION, [119](#)
SYNCIN_INVERT, [119](#)
SYNCOUT_ENABLED, [119](#)
SYNCOUT_IN_STEPS, [119](#)
SYNCOUT_INVERT, [119](#)
SYNCOUT_ONPERIOD, [119](#)
SYNCOUT_ONSTART, [119](#)
SYNCOUT_ONSTOP, [119](#)
SYNCOUT_STATE, [119](#)
service_command_updf, [147](#)
set_accessories_settings, [147](#)
set_bindy_key, [147](#)
set_brake_settings, [147](#)
set_calibration_settings, [148](#)
set_control_settings, [148](#)
set_control_settings_calb, [148](#)
set_controller_name, [149](#)
set_correction_table, [149](#)
set_ctp_settings, [149](#)
set_debug_write, [150](#)
set_edges_settings, [150](#)
set_edges_settings_calb, [150](#)
set_emf_settings, [151](#)
set_encoder_information, [151](#)
set_encoder_settings, [151](#)
set_engine_advanced_setup, [152](#)
set_engine_settings, [152](#)
set_engine_settings_calb, [152](#)
set_entype_settings, [153](#)
set_extended_settings, [153](#)
set_extio_settings, [153](#)
set_feedback_settings, [153](#)
set_gear_information, [154](#)
set_gear_settings, [154](#)
set_hallsensor_information, [154](#)
set_hallsensor_settings, [154](#)
set_home_settings, [155](#)
set_home_settings_calb, [155](#)

- set_joystick_settings, [155](#)
- set_logging_callback, [156](#)
- set_motor_information, [156](#)
- set_motor_settings, [156](#)
- set_move_settings, [156](#)
- set_move_settings_calb, [157](#)
- set_network_settings, [157](#)
- set_nonvolatile_memory, [157](#)
- set_password_settings, [157](#)
- set_pid_settings, [158](#)
- set_position, [158](#)
- set_position_calb, [158](#)
- set_power_settings, [158](#)
- set_secure_settings, [159](#)
- set_serial_number, [159](#)
- set_stage_information, [159](#)
- set_stage_name, [159](#)
- set_stage_settings, [160](#)
- set_sync_in_settings, [160](#)
- set_sync_in_settings_calb, [160](#)
- set_sync_out_settings, [161](#)
- set_sync_out_settings_calb, [161](#)
- set_uart_settings, [161](#)
- UART_PARITY_BITS, [119](#)
- WIND_A_STATE_OK, [120](#)
- WIND_B_STATE_OK, [120](#)
- write_key, [162](#)
- XIMC_API, [120](#)
- ximc_fix_usbser_sys, [162](#)
- ximc_version, [162](#)
- ximc_fix_usbser_sys
 - ximc.h, [162](#)
- ximc_version
 - ximc.h, [162](#)