

cr•ssc•ntr•l

CCAux
1.4.0.0

Tue Aug 28 2012

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Class Index	2
2.1	Class List	2
3	File Index	3
3.1	File List	3
4	Namespace Documentation	4
4.1	CrossControl Namespace Reference	4
4.1.1	Typedef Documentation	8
4.1.1.1	VersionType	8
4.1.2	Enumeration Type Documentation	8
4.1.2.1	ButtonPowerTransitionStatus	8
4.1.2.2	CalibrationConfigParam	9
4.1.2.3	CalibrationModeSettings	9
4.1.2.4	CanFrameType	9
4.1.2.5	CCAuxColor	9
4.1.2.6	CCStatus	10
4.1.2.7	DeInterlaceMode	10
4.1.2.8	eErr	10
4.1.2.9	hwErrorStatusCodes	11
4.1.2.10	JidaSensorType	15
4.1.2.11	LightSensorOperationRange	15
4.1.2.12	LightSensorSamplingMode	16
4.1.2.13	PowerAction	16
4.1.2.14	shutDownReasonCodes	16
4.1.2.15	startupReasonCodes	16
4.1.2.16	TouchScreenModeSettings	17
4.1.2.17	TriggerConf	17
4.1.2.18	TSAAdvancedSettingsParameter	17
4.1.2.19	UpgradeAction	18
4.1.2.20	VideoChannel	18
4.1.2.21	videoStandard	19
4.1.2.22	VoltageEnum	19
4.1.3	Function Documentation	19
4.1.3.1	GetErrorStringA	19
4.1.3.2	GetErrorStringW	20

4.1.3.3	GetHwErrorStatusStringA	20
4.1.3.4	GetHwErrorStatusStringW	20
4.1.3.5	GetStartupReasonStringA	21
4.1.3.6	GetStartupReasonStringW	21
4.1.4	Variable Documentation	21
4.1.4.1	DigitalIn_1	21
4.1.4.2	DigitalIn_2	21
4.1.4.3	DigitalIn_3	21
4.1.4.4	DigitalIn_4	21
4.1.4.5	Video1Conf	21
4.1.4.6	Video2Conf	22
4.1.4.7	Video3Conf	22
4.1.4.8	Video4Conf	22
5	Class Documentation	23
5.1	CrossControl::About Struct Reference	23
5.1.1	Detailed Description	24
5.1.2	Member Function Documentation	26
5.1.2.1	getAddOnHWversion	26
5.1.2.2	getAddOnManufacturingDate	26
5.1.2.3	getAddOnPCBArt	27
5.1.2.4	getAddOnPCBSerial	27
5.1.2.5	getIsBTMounted	28
5.1.2.6	getIsDisplayAvailable	28
5.1.2.7	getIsGPRSMounted	29
5.1.2.8	getIsGPSMounted	29
5.1.2.9	getIsTouchScreenAvailable	29
5.1.2.10	getIsWLANMounted	30
5.1.2.11	getMainHWversion	30
5.1.2.12	getMainManufacturingDate	31
5.1.2.13	getMainPCBArt	31
5.1.2.14	getMainPCBSerial	31
5.1.2.15	getMainProdArtNr	32
5.1.2.16	getMainProdRev	32
5.1.2.17	getNrOfCANConnections	33
5.1.2.18	getNrOfDigIOConnections	33
5.1.2.19	getNrOfETHConnections	34
5.1.2.20	getNrOfSerialConnections	34
5.1.2.21	getNrOfUSBConnections	34
5.1.2.22	getNrOfVideoConnections	35
5.1.2.23	getUnitSerial	35
5.1.2.24	Release	36
5.2	CrossControl::Adc Struct Reference	36
5.2.1	Detailed Description	36
5.2.2	Member Function Documentation	37
5.2.2.1	getVoltage	37
5.2.2.2	Release	38
5.3	CrossControl::AuxVersion Struct Reference	38
5.3.1	Detailed Description	39
5.3.2	Member Function Documentation	41

5.3.2.1	getCCAuxDrvVersion	41
5.3.2.2	getCCAuxVersion	42
5.3.2.3	getFPGAVersion	42
5.3.2.4	getFrontVersion	43
5.3.2.5	getOSVersion	44
5.3.2.6	getSSVersion	44
5.3.2.7	Release	45
5.4	CrossControl::Backlight Struct Reference	45
5.4.1	Detailed Description	46
5.4.2	Member Function Documentation	47
5.4.2.1	getAutomaticBLFilter	47
5.4.2.2	getAutomaticBLParams	48
5.4.2.3	getAutomaticBLStatus	48
5.4.2.4	getIntensity	48
5.4.2.5	getLedDimming	49
5.4.2.6	getStatus	49
5.4.2.7	Release	50
5.4.2.8	setAutomaticBLFilter	50
5.4.2.9	setAutomaticBLParams	51
5.4.2.10	setIntensity	51
5.4.2.11	setLedDimming	52
5.4.2.12	startAutomaticBL	52
5.4.2.13	stopAutomaticBL	52
5.5	CrossControl::Buzzer Struct Reference	52
5.5.1	Detailed Description	53
5.5.2	Member Function Documentation	54
5.5.2.1	buzze	54
5.5.2.2	getFrequency	55
5.5.2.3	getTrigger	55
5.5.2.4	getVolume	56
5.5.2.5	Release	56
5.5.2.6	setFrequency	56
5.5.2.7	setTrigger	57
5.5.2.8	setVolume	57
5.6	CrossControl::BuzzerSetup Struct Reference	58
5.6.1	Member Data Documentation	58
5.6.1.1	frequency	58
5.6.1.2	volume	58
5.7	CrossControl::CanSetting Struct Reference	58
5.7.1	Detailed Description	59
5.7.2	Member Function Documentation	60
5.7.2.1	getBaudrate	60
5.7.2.2	getFrameType	61
5.7.2.3	Release	61
5.7.2.4	setBaudrate	61
5.7.2.5	setFrameType	62
5.8	CrossControl::Config Struct Reference	62
5.8.1	Detailed Description	63
5.8.2	Member Function Documentation	64
5.8.2.1	getCanStartupPowerConfig	64

5.8.2.2	getExtFanStartupPowerConfig	64
5.8.2.3	getExtOnOffSigTrigTime	65
5.8.2.4	getFrontBtnTrigTime	65
5.8.2.5	getHeatingTempLimit	65
5.8.2.6	getLongButtonPressAction	66
5.8.2.7	getOnOffSigAction	66
5.8.2.8	getPowerOnStartup	66
5.8.2.9	getShortButtonPressAction	67
5.8.2.10	getStartupTriggerConfig	67
5.8.2.11	getStartupVoltageConfig	68
5.8.2.12	getSuspendMaxTime	68
5.8.2.13	getVideoStartupPowerConfig	68
5.8.2.14	Release	69
5.8.2.15	setCanStartupPowerConfig	69
5.8.2.16	setExtFanStartupPowerConfig	69
5.8.2.17	setExtOnOffSigTrigTime	70
5.8.2.18	setFrontBtnTrigTime	70
5.8.2.19	setHeatingTempLimit	70
5.8.2.20	setLongButtonPressAction	71
5.8.2.21	setOnOffSigAction	71
5.8.2.22	setPowerOnStartup	71
5.8.2.23	setShortButtonPressAction	72
5.8.2.24	setStartupTriggerConfig	72
5.8.2.25	setStartupVoltageConfig	73
5.8.2.26	setSuspendMaxTime	73
5.8.2.27	setVideoStartupPowerConfig	73
5.9	CrossControl::Diagnostic Struct Reference	74
5.9.1	Detailed Description	74
5.9.2	Member Function Documentation	76
5.9.2.1	clearHwErrorStatus	76
5.9.2.2	getHwErrorStatus	76
5.9.2.3	getMinMaxTemp	76
5.9.2.4	getPCBTemp	77
5.9.2.5	getPMTemp	77
5.9.2.6	getPowerCycles	77
5.9.2.7	getShutDownReason	78
5.9.2.8	getSSTemp	78
5.9.2.9	getStartupReason	78
5.9.2.10	getTimer	79
5.9.2.11	Release	79
5.10	CrossControl::DigIO Struct Reference	79
5.10.1	Detailed Description	80
5.10.2	Member Function Documentation	81
5.10.2.1	getDigIO	81
5.10.2.2	getDigIODir	81
5.10.2.3	Release	82
5.10.2.4	setDigIO	82
5.10.2.5	setDigIODir	82
5.11	CrossControl::FirmwareUpgrade Struct Reference	83
5.11.1	Detailed Description	83

5.11.2	Member Function Documentation	86
5.11.2.1	getUpgradeStatus	86
5.11.2.2	Release	86
5.11.2.3	shutDown	87
5.11.2.4	startFpgaUpgrade	87
5.11.2.5	startFpgaVerification	88
5.11.2.6	startFrontUpgrade	89
5.11.2.7	startFrontVerification	90
5.11.2.8	startSSUpgrade	91
5.11.2.9	startSSVerification	92
5.12	CrossControl::FrontLED Struct Reference	93
5.12.1	Detailed Description	94
5.12.2	Member Function Documentation	95
5.12.2.1	getColor	95
5.12.2.2	getColor	96
5.12.2.3	getEnabledDuringStartup	96
5.12.2.4	getIdleTime	96
5.12.2.5	getNrOfPulses	97
5.12.2.6	getOffTime	97
5.12.2.7	getOnTime	97
5.12.2.8	getSignal	98
5.12.2.9	Release	98
5.12.2.10	setColor	98
5.12.2.11	setColor	99
5.12.2.12	setEnabledDuringStartup	99
5.12.2.13	setIdleTime	100
5.12.2.14	setNrOfPulses	100
5.12.2.15	setOff	100
5.12.2.16	setOffTime	100
5.12.2.17	setOnTime	101
5.12.2.18	setSignal	101
5.13	CrossControl::LedColorMixType Struct Reference	102
5.13.1	Member Data Documentation	102
5.13.1.1	blue	102
5.13.1.2	green	102
5.13.1.3	red	102
5.14	CrossControl::LedTimingType Struct Reference	102
5.14.1	Member Data Documentation	103
5.14.1.1	idleTime	103
5.14.1.2	nrOfPulses	103
5.14.1.3	offTime	103
5.14.1.4	onTime	103
5.15	CrossControl::Lightsensor Struct Reference	103
5.15.1	Detailed Description	104
5.15.2	Member Function Documentation	105
5.15.2.1	getAverageIlluminance	105
5.15.2.2	getIlluminance	105
5.15.2.3	getIlluminance	106
5.15.2.4	getOperatingRange	106
5.15.2.5	Release	107

5.15.2.6	setOperatingRange	107
5.15.2.7	startAverageCalc	107
5.15.2.8	stopAverageCalc	108
5.16	CrossControl::Power Struct Reference	108
5.16.1	Detailed Description	109
5.16.2	Member Function Documentation	110
5.16.2.1	ackPowerRequest	110
5.16.2.2	getBLPowerStatus	110
5.16.2.3	getButtonPowerTransitionStatus	111
5.16.2.4	getCanPowerStatus	111
5.16.2.5	getExtFanPowerStatus	112
5.16.2.6	getVideoPowerStatus	112
5.16.2.7	Release	112
5.16.2.8	setBLPowerStatus	113
5.16.2.9	setCanPowerStatus	113
5.16.2.10	setExtFanPowerStatus	114
5.16.2.11	setVideoPowerStatus	114
5.17	CrossControl::received_video Struct Reference	114
5.17.1	Member Data Documentation	115
5.17.1.1	received_framerate	115
5.17.1.2	received_height	115
5.17.1.3	received_width	115
5.18	CrossControl::Telematics Struct Reference	115
5.18.1	Detailed Description	115
5.18.2	Member Function Documentation	119
5.18.2.1	getBTPowerStatus	119
5.18.2.2	getBTStartUpPowerStatus	119
5.18.2.3	getGPRSPowerStatus	120
5.18.2.4	getGPRSStartUpPowerStatus	121
5.18.2.5	getGPSAntennaStatus	121
5.18.2.6	getGPSPowerStatus	122
5.18.2.7	getGPSStartUpPowerStatus	122
5.18.2.8	getTelematicsAvailable	123
5.18.2.9	getWLANPowerStatus	124
5.18.2.10	getWLANStartUpPowerStatus	124
5.18.2.11	Release	125
5.18.2.12	setBTPowerStatus	125
5.18.2.13	setBTStartUpPowerStatus	125
5.18.2.14	setGPRSPowerStatus	126
5.18.2.15	setGPRSStartUpPowerStatus	126
5.18.2.16	setGPSPowerStatus	126
5.18.2.17	setGPSStartUpPowerStatus	127
5.18.2.18	setWLANPowerStatus	127
5.18.2.19	setWLANStartUpPowerStatus	127
5.19	CrossControl::TimerType Struct Reference	128
5.19.1	Detailed Description	128
5.19.2	Member Data Documentation	128
5.19.2.1	Above80RunTime	128
5.19.2.2	RunTime40_60	128
5.19.2.3	RunTime60_70	128

5.19.2.4	RunTime70_80	128
5.19.2.5	TotHeatTime	128
5.19.2.6	TotRunTime	129
5.19.2.7	TotSuspTime	129
5.20	CrossControl::TouchScreen Struct Reference	129
5.20.1	Detailed Description	129
5.20.2	Member Function Documentation	130
5.20.2.1	getAdvancedSetting	131
5.20.2.2	getMode	131
5.20.2.3	getMouseRightClickTime	132
5.20.2.4	Release	132
5.20.2.5	setAdvancedSetting	133
5.20.2.6	setMode	133
5.20.2.7	setMouseRightClickTime	133
5.21	CrossControl::TouchScreenCalib Struct Reference	134
5.21.1	Detailed Description	134
5.21.2	Member Function Documentation	134
5.21.2.1	CheckCalibrationPointFinished	134
5.21.2.2	GetConfigParam	135
5.21.2.3	GetMode	135
5.21.2.4	Release	135
5.21.2.5	SetCalibrationPoint	135
5.21.2.6	SetConfigParam	136
5.21.2.7	SetMode	136
5.22	CrossControl::UpgradeStatus Struct Reference	136
5.22.1	Detailed Description	137
5.22.2	Member Data Documentation	137
5.22.2.1	currentAction	137
5.22.2.2	errorCode	137
5.22.2.3	percent	137
5.23	CrossControl::version_info Struct Reference	137
5.23.1	Member Data Documentation	137
5.23.1.1	build	137
5.23.1.2	major	137
5.23.1.3	minor	138
5.23.1.4	release	138
5.24	CrossControl::Video Struct Reference	138
5.24.1	Detailed Description	139
5.24.2	Member Function Documentation	139
5.24.2.1	activateSnapshot	139
5.24.2.2	createBitmap	139
5.24.2.3	freeBmpBuffer	140
5.24.2.4	getActiveChannel	140
5.24.2.5	getColorKeys	141
5.24.2.6	getCropping	141
5.24.2.7	getDecoderReg	141
5.24.2.8	getDeInterlaceMode	142
5.24.2.9	getMirroring	142
5.24.2.10	getRawImage	142
5.24.2.11	getScaling	143

5.24.2.12	getStatus	143
5.24.2.13	getVideoArea	143
5.24.2.14	getVideoStandard	144
5.24.2.15	init	144
5.24.2.16	minimize	144
5.24.2.17	Release	145
5.24.2.18	restore	145
5.24.2.19	setActiveChannel	145
5.24.2.20	setColorKeys	145
5.24.2.21	setCropping	146
5.24.2.22	setDecoderReg	146
5.24.2.23	setDeInterlaceMode	146
5.24.2.24	setMirroring	147
5.24.2.25	setScaling	147
5.24.2.26	setVideoArea	147
5.24.2.27	showVideo	148
5.24.2.28	takeSnapshot	148
5.24.2.29	takeSnapshotBmp	148
5.24.2.30	takeSnapshotRaw	149
5.25	CrossControl::video_dec_command Struct Reference	149
5.25.1	Member Data Documentation	150
5.25.1.1	decoder_register	150
5.25.1.2	register_value	150
6	File Documentation	151
6.1	fixedIncludeFiles/About.h File Reference	151
6.1.1	Typedef Documentation	151
6.1.1.1	ABOUTHANDLE	151
6.1.2	Function Documentation	151
6.1.2.1	GetAbout	151
6.2	fixedIncludeFiles/Adc.h File Reference	152
6.2.1	Typedef Documentation	153
6.2.1.1	ADCHANDLE	153
6.2.2	Function Documentation	153
6.2.2.1	GetAdc	153
6.3	fixedIncludeFiles/AuxVersion.h File Reference	153
6.3.1	Typedef Documentation	154
6.3.1.1	AUXVERSIONHANDLE	154
6.3.2	Function Documentation	154
6.3.2.1	GetAuxVersion	154
6.4	fixedIncludeFiles/Backlight.h File Reference	154
6.4.1	Typedef Documentation	155
6.4.1.1	BACKLIGHTHANDLE	155
6.4.2	Function Documentation	155
6.4.2.1	GetBacklight	155
6.5	fixedIncludeFiles/Buzzer.h File Reference	155
6.5.1	Typedef Documentation	156
6.5.1.1	BUZZERHANDLE	156
6.5.2	Function Documentation	156
6.5.2.1	GetBuzzer	156

6.6	fixedIncludeFiles/CanSetting.h File Reference	156
6.6.1	Typedef Documentation	157
6.6.1.1	CANSETTINGHANDLE	157
6.6.2	Function Documentation	157
6.6.2.1	GetCanSetting	157
6.7	fixedIncludeFiles/CCAuxErrors.h File Reference	157
6.8	fixedIncludeFiles/CCAuxTypes.h File Reference	158
6.9	fixedIncludeFiles/CCPlatform.h File Reference	159
6.10	fixedIncludeFiles/Config.h File Reference	160
6.10.1	Typedef Documentation	160
6.10.1.1	CONFIGHANDLE	160
6.10.2	Function Documentation	160
6.10.2.1	GetConfig	160
6.11	fixedIncludeFiles/Diagnostic.h File Reference	161
6.11.1	Typedef Documentation	161
6.11.1.1	DIAGNOSTICHANDLE	161
6.11.2	Function Documentation	161
6.11.2.1	GetDiagnostic	161
6.12	fixedIncludeFiles/DiagnosticCodes.h File Reference	162
6.13	fixedIncludeFiles/DigIO.h File Reference	164
6.13.1	Typedef Documentation	165
6.13.1.1	DIGIOHANDLE	165
6.13.2	Function Documentation	165
6.13.2.1	GetDigIO	165
6.14	fixedIncludeFiles/FirmwareUpgrade.h File Reference	165
6.14.1	Typedef Documentation	166
6.14.1.1	FIRMWAREUPGHANDLE	166
6.14.2	Function Documentation	166
6.14.2.1	GetFirmwareUpgrade	166
6.15	fixedIncludeFiles/FrontLED.h File Reference	166
6.15.1	Typedef Documentation	166
6.15.1.1	FRONTLEDHANDLE	166
6.15.2	Function Documentation	167
6.15.2.1	GetFrontLED	167
6.16	fixedIncludeFiles/Lightsensor.h File Reference	167
6.16.1	Typedef Documentation	167
6.16.1.1	LIGHTSENSORHANDLE	167
6.16.2	Function Documentation	167
6.16.2.1	GetLightsensor	168
6.17	fixedIncludeFiles/Power.h File Reference	168
6.17.1	Typedef Documentation	168
6.17.1.1	POWERHANDLE	168
6.17.2	Function Documentation	168
6.17.2.1	GetPower	169
6.18	fixedIncludeFiles/Telematics.h File Reference	169
6.18.1	Typedef Documentation	169
6.18.1.1	TELEMATICSHANDLE	169
6.18.2	Function Documentation	169
6.18.2.1	GetTelematics	170
6.19	fixedIncludeFiles/TouchScreen.h File Reference	170

6.19.1	Typedef Documentation	171
6.19.1.1	TOUCHSCREENHANDLE	171
6.19.2	Function Documentation	171
6.19.2.1	GetTouchScreen	171
6.20	fixedIncludeFiles/TouchScreenCalib.h File Reference	171
6.20.1	Typedef Documentation	172
6.20.1.1	TOUCHSCREENCALIBHANDLE	172
6.20.2	Function Documentation	172
6.20.2.1	GetTouchScreenCalib	172
6.21	fixedIncludeFiles/Video.h File Reference	172
6.21.1	Typedef Documentation	173
6.21.1.1	VIDEOHANDLE	173
6.21.2	Function Documentation	173
6.21.2.1	GetVideo	173

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

CrossControl	4
--	---

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CrossControl::About	23
CrossControl::Adc	36
CrossControl::AuxVersion	38
CrossControl::Backlight	45
CrossControl::Buzzer	52
CrossControl::BuzzerSetup	58
CrossControl::CanSetting	58
CrossControl::Config	62
CrossControl::Diagnostic	74
CrossControl::DigIO	79
CrossControl::FirmwareUpgrade	83
CrossControl::FrontLED	93
CrossControl::LedColorMixType	102
CrossControl::LedTimingType	102
CrossControl::LightSensor	103
CrossControl::Power	108
CrossControl::received_video	114
CrossControl::Telematics	115
CrossControl::TimerType	128
CrossControl::TouchScreen	129
CrossControl::TouchScreenCalib	134
CrossControl::UpgradeStatus	136
CrossControl::version_info	137
CrossControl::Video	138
CrossControl::video_dec_command	149

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

fixedIncludeFiles/About.h	151
fixedIncludeFiles/Adc.h	152
fixedIncludeFiles/AuxVersion.h	153
fixedIncludeFiles/Backlight.h	154
fixedIncludeFiles/Buzzer.h	155
fixedIncludeFiles/CanSetting.h	156
fixedIncludeFiles/CCAuxErrors.h	157
fixedIncludeFiles/CCAuxTypes.h	158
fixedIncludeFiles/CCPlatform.h	159
fixedIncludeFiles/Config.h	160
fixedIncludeFiles/Diagnostic.h	161
fixedIncludeFiles/DiagnosticCodes.h	162
fixedIncludeFiles/DigIO.h	164
fixedIncludeFiles/FirmwareUpgrade.h	165
fixedIncludeFiles/FrontLED.h	166
fixedIncludeFiles/Lightsensor.h	167
fixedIncludeFiles/Power.h	168
fixedIncludeFiles/Telematics.h	169
fixedIncludeFiles/TouchScreen.h	170
fixedIncludeFiles/TouchScreenCalib.h	171
fixedIncludeFiles/Video.h	172

Chapter 4

Namespace Documentation

4.1 CrossControl Namespace Reference

Classes

- struct [About](#)
- struct [Adc](#)
- struct [AuxVersion](#)
- struct [Backlight](#)
- struct [Buzzer](#)
- struct [CanSetting](#)
- struct [received_video](#)
- struct [video_dec_command](#)
- struct [version_info](#)
- struct [BuzzerSetup](#)
- struct [LedTimingType](#)
- struct [LedColorMixType](#)
- struct [TimerType](#)
- struct [UpgradeStatus](#)
- struct [Config](#)
- struct [Diagnostic](#)
- struct [DigIO](#)
- struct [FirmwareUpgrade](#)
- struct [FrontLED](#)
- struct [Lightsensor](#)
- struct [Power](#)
- struct [Telematics](#)
- struct [TouchScreen](#)
- struct [TouchScreenCalib](#)
- struct [Video](#)

Typedefs

- typedef struct `version_info` `VersionType`

Enumerations

- enum `VoltageEnum` { `VOLTAGE_24VIN` = 0, `VOLTAGE_24V`, `VOLTAGE_12V`, `VOLTAGE_12VID`, `VOLTAGE_5V`, `VOLTAGE_3V3`, `VOLTAGE_VTFT`, `VOLTAGE_5VSTB`, `VOLTAGE_1V9`, `VOLTAGE_1V8`, `VOLTAGE_1V5`, `VOLTAGE_1V2`, `VOLTAGE_1V05`, `VOLTAGE_1V0`, `VOLTAGE_0V9`, `VOLTAGE_VREF_INT` }
- enum `LightSensorOperationRange` { `RangeStandard` = 0, `RangeExtended` = 1 }
- enum `LightSensorSamplingMode` { `SamplingModeStandard` = 0, `SamplingModeExtended`, `SamplingModeAuto` }
- enum `CCStatus` { `Disabled` = 0, `Enabled` = 1 }
- enum `eErr` { `ERR_SUCCESS` = 0, `ERR_OPEN_FAILED` = 1, `ERR_NOT_SUPPORTED` = 2, `ERR_UNKNOWN_FEATURE` = 3, `ERR_DATATYPE_MISMATCH` = 4, `ERR_CODE_NOT_EXIST` = 5, `ERR_BUFFER_SIZE` = 6, `ERR_IOCTL_FAILED` = 7, `ERR_INVALID_DATA` = 8, `ERR_INVALID_PARAMETER` = 9, `ERR_CREATE_THREAD` = 10, `ERR_IN_PROGRESS` = 11, `ERR_CHECKSUM` = 12, `ERR_INIT_FAILED` = 13, `ERR_VERIFY_FAILED` = 14, `ERR_DEVICE_READ_DATA_FAILED` = 15, `ERR_DEVICE_WRITE_DATA_FAILED` = 16, `ERR_COMMAND_FAILED` = 17, `ERR_EEPROM` = 18, `ERR_JIDA_TEMP` = 19, `ERR_AVERAGE_CALC_STARTED` = 20, `ERR_NOT_RUNNING` = 21, `ERR_I2C_EXPANDER_READ_FAILED` = 22, `ERR_I2C_EXPANDER_WRITE_FAILED` = 23, `ERR_I2C_EXPANDER_INIT_FAILED` = 24, `ERR_NEWER_SS_VERSION_REQUIRED` = 25, `ERR_NEWER_FPGA_VERSION_REQUIRED` = 26, `ERR_NEWER_FRONT_VERSION_REQUIRED` = 27, `ERR_TELEMATICS_GPRS_NOT_AVAILABLE` = 28, `ERR_TELEMATICS_WLAN_NOT_AVAILABLE` = 29, `ERR_TELEMATICS_BT_NOT_AVAILABLE` = 30, `ERR_TELEMATICS_GPS_NOT_AVAILABLE` = 31, `ERR_MEM_ALLOC_FAIL` = 32 }
- enum `DeInterlaceMode` { `DeInterlace_Even` = 0, `DeInterlace_Odd` = 1, `DeInterlace_BOB` = 2 }
- enum `VideoChannel` { `Analog_Channel_1` = 0, `Analog_Channel_2` = 1, `Analog_Channel_3` = 2, `Analog_Channel_4` = 3 }
- enum `videoStandard` { `STD_M_J_NTSC` = 0, `STD_B_D_G_H_I_N_PAL` = 1, `STD_M_PAL` = 2, `STD_PAL` = 3, `STD_NTSC` = 4, `STD_SECAM` = 5 }
- enum `CanFrameType` { `FrameStandard`, `FrameExtended`, `FrameStandardExtended` }
- enum `TriggerConf` { `Front_Button_Enabled` = 1, `OnOff_Signal_Enabled` = 2, `Both_Button_And_Signal_Enabled` = 3 }
- enum `PowerAction` { `NoAction` = 0, `ActionSuspend` = 1, `ActionShutDown` = 2 }
- enum `ButtonPowerTransitionStatus` { `BPTS_No_Change` = 0, `BPTS_ShutDown` = 1, `BPTS_Suspend` = 2, `BPTS_Restart` = 3, `BPTS_BtnPressed` = 4, `BPTS_BtnPressedLong` = 5, `BPTS_SignalOff` = 6 }
- enum `JidaSensorType` { `TEMP_CPU` = 0, `TEMP_BOX` = 1, `TEMP_ENV` = 2, `TEMP_BOARD` = 3, `TEMP_BACKPLANE` = 4, `TEMP_CHIPSETS` = 5, `TEMP_VIDEO` = 6, `TEMP_OTHER` = 7 }

- enum UpgradeAction { UPGRADE_INIT, UPGRADE_PREP_COM, UPGRADE_READING_FILE, UPGRADE_CONVERTING_FILE, UPGRADE_FLASHING, UPGRADE_VERIFYING, UPGRADE_COMPLETE, UPGRADE_COMPLETE_WITH_ERRORS }
- enum CCAuxColor { RED = 0, GREEN, BLUE, CYAN, MAGENTA, YELLOW, UNDEFINED_COLOR }
- enum startupReasonCodes { startupReasonCodeUndefined = 0x0000, startupReasonCodeButtonPress = 0x0055, startupReasonCodeExtCtrl = 0x00AA, startupReasonCodeMPRestart = 0x00F0, startupReasonCodePowerOnStartup = 0x000F }
- enum shutDownReasonCodes { shutdownReasonCodeNoError = 0x001F }
- enum hwErrorStatusCodes { errCodeNoErr = 0, errCodeFPGACONFReadErr = 1, errCodeFPGACONFUnexpVal = 2, errCodeCBRESETReadErr = 3, errCodeSUS3ReadErr = 4, errCodeSUS4ReadErr = 5, errCodeSUS5ReadErr = 6, errCodePG5VSTBYReadErr = 7, errCodePG5VSTBYUnexpVal = 8, errCodeCANPWROKReadErr = 9, errCodeVIDPWROKReadErr = 10, errCodeLVDSBLENReadErr = 11, errCodeLVDSVDDENReadErr = 12, errCodeEXTCTRLONReadErr = 13, errCodeFPBTNONReadErr = 14, errCode24VReadErr = 15, errCode24VOutOfLimits = 16, errCode24VINReadErr = 17, errCode24VINOutOfLimits = 18, errCode12VReadErr = 19, errCode12VOutOfLimits = 20, errCode12VVIDEOREadErr = 21, errCode12VVIDEOOutOfLimits = 22, errCode5VSTBYReadErr = 23, errCode5VSTBYOutOfLimits = 24, errCode5VReadErr = 25, errCode5VOutOfLimits = 26, errCode3V3ReadErr = 27, errCode3V3OutOfLimits = 28, errCodeTFTVOLReadErr = 29, errCodeTFTVOLOutOfLimits = 30, errCode1V9ReadErr = 31, errCode1V9OutOfLimits = 32, errCode1V8ReadErr = 33, errCode1V8OutOfLimits = 34, errCode1V5ReadErr = 35, errCode1V5OutOfLimits = 36, errCode1V2ReadErr = 37, errCode1V2OutOfLimits = 38, errCode1V05ReadErr = 39, errCode1V05OutOfLimits = 40, errCode1V0ReadErr = 41, errCode1V0OutOfLimits = 42, errCode0V9ReadErr = 43, errCode0V9OutOfLimits = 44, errCodeI2CTEMPReadErr = 45, errCodeI2CTEMPOOutOfLimits = 46, errCodeSTM32TEMPReadErr = 47, errCodeSTM32TEMPOOutOfLimits = 48, errCodeBLTYPEUnexpEEPROMVal = 49, errCodeFPBTNUnexpEEPROMVal = 50, errCodeEXTCTRLUnexpEEPROMVal = 51, errCodeLowRange24VUnexpEEPROMVal = 52, errCodeSuspToRAMUnexpEEPROMVal = 53, errCodeCANPWRUnexpEEPROMVal = 54, errCodeVID1PWRUnexpEEPROMVal = 55, errCodeVID2PWRUnexpEEPROMVal = 56, errCodeVID3PWRUnexpEEPROMVal = 57, errCodeVID4PWRUnexpEEPROMVal = 58, errCodeEXTFANUnexpEEPROMVal = 59, errCodeLEDUnexpEEPROMVal = 60, errCodeUnitTypeUnexpEEPROMVal = 61, errCodeBLTYPEReadErrEEPROM = 62, errCodeFPBTNReadErrEEPROM = 63, errCodeEXTCTRLReadErrEEPROM = 64, errCodeMaxSuspTimeReadErrEEPROM = 65, errCodeLowRange24VReadErrEEPROM = 66, errCodeSuspToRAMReadErrEEPROM = 67, errCodeCANPWRReadErrEEPROM = 68, errCodeVID1PWRReadErrEEPROM = 69, errCodeVID2PWRReadErrEEPROM = 70, errCodeVID3PWRReadErrEEPROM = 71, errCodeVID4PWRReadErrEEPROM = 72, errCodeEXTFANReadErrEEPROM = 73, errCodeLEDReadErrEEPROM = 74, errCodeUnitTypeReadErrEEPROM = 75, errCodeRCCInit = 76, errCodeDriverInit = 77, errCodeSetSUPPLYRESET = 78, errCodeRelSUPPLYRESET = 79, errCodeSetSYSRESET = 80, errCodeRelSYSRESET = 81, errCodeSetPWRBTN = 82, errCodeRelPWRBTN = 83, errCodeOnBL = 84, errCodeOffBL

- = 85, [errCodeEXTFANOn](#) = 86, [errCodeEXTFANOff](#) = 87, [errCodePWRENOn](#) = 88, [errCodePWRENOff](#) = 89, [errCodeMPPWRENOn](#) = 90, [errCodeMPPWRENOff](#) = 91, [errCodeCANPWRENOn](#) = 92, [errCodeCANPWRENOff](#) = 93, [errCodeVID1PWRENOn](#) = 94, [errCodeVID1PWRENOff](#) = 95, [errCodeVID2PWRENOn](#) = 96, [errCodeVID2PWRENOff](#) = 97, [errCodeVID3PWRENOn](#) = 98, [errCodeVID3PWRENOff](#) = 99, [errCodeVID4PWRENOn](#) = 100, [errCodeVID4PWRENOff](#) = 101, [errCodeHEATACTOn](#) = 102, [errCodeHEATACTOff](#) = 103, [errCodeSetLEDCol](#) = 104, [errCodeSetLEDFreq](#) = 105, [errCodeManageLED](#) = 106, [errCodeManageCANPwr](#) = 107, [errCodeManageMPPwv](#) = 108, [errCodeManageVidPwr](#) = 109, [errCodeManagePowSup](#) = 110, [errCodeManageReset](#) = 111, [errCodeSSState](#) = 112, [errCodeVarWrapAround](#) = 113, [errCodeFPBTNUnexpVal](#) = 114, [errCodeEXTCTRLUnexpVal](#) = 115, [errCodeMAINPWROKReadErr](#) = 116, [errCodeFRONTSPAREReadErr](#) = 117, [errCodeTIMERReadErr](#) = 118, [errCodeManageDiagnostics](#) = 119, [errCodeFPBTNTimOutReadErrEEPROM](#) = 120, [errCodeEXTCTRLTimOutReadErrEEPROM](#) = 121, [errCodeFPBTNTAndExtCtrlDisabled](#) = 122, [errCodeSWVerReadErr](#) = 123, [errCodeSWVerWriteErr](#) = 124, [errCodeManageActDeAct](#) = 125, [errCodeTickTimeOutTimer](#) = 126, [errCodeOperateModeStateError](#) = 127, [errCodeHeatingTempReadErrEEPROM](#) = 128, [errCodeMPFailedStart](#) = 129, [errCodeReadErrEEPROM](#) = 130, [errCodeTimeOutWaitingForVoltages](#) = 131, [errCodeMAX](#) }
- enum [TouchScreenModeSettings](#) { [MOUSE_NEXT_BOOT](#) = 0, [TOUCH_NEXT_BOOT](#) = 1, [MOUSE_NOW](#) = 2, [TOUCH_NOW](#) = 3 }
 - enum [TSAdvancedSettingsParameter](#) { [TS_RIGHT_CLICK_TIME](#) = 0, [TS_LOW_LEVEL](#) = 1, [TS_UNTOUCHLEVEL](#) = 2, [TS_DEBOUNCE_TIME](#) = 3, [TS_DEBOUNCE_TIMEOUT_TIME](#) = 4, [TS_DOUBLECLICK_MAX_CLICK_TIME](#) = 5, [TS_DOUBLE_CLICK_TIME](#) = 6, [TS_MAX_RIGHTCLICK_DISTANCE](#) = 7, [TS_USE_DEJITTER](#) = 8, [TS_CALIBTATION_WIDTH](#) = 9, [TS_CALIBRATION_MEASUREMENTS](#) = 10, [TS_RESTORE_DEFAULT_SETTINGS](#) = 11 }
 - enum [CalibrationModeSettings](#) { [MODE_UNKNOWN](#) = 0, [MODE_NORMAL](#) = 1, [MODE_CALIBRATION_5P](#) = 2, [MODE_CALIBRATION_9P](#) = 3, [MODE_CALIBRATION_13P](#) = 4 }
 - enum [CalibrationConfigParam](#) { [CONFIG_CALIBRATION_WITH](#) = 0, [CONFIG_CALIBRATION_MEASUREMENTS](#) = 1, [CONFIG_5P_CALIBRATION_POINT_BORDER](#) = 2, [CONFIG_13P_CALIBRATION_POINT_BORDER](#) = 3, [CONFIG_13P_CALIBRATION_TRANSITION_MIN](#) = 4, [CONFIG_13P_CALIBRATION_TRANSITION_MAX](#) = 5 }

Functions

- [EXTERN_C CCAUXDLL_API char const *CCAUXDLL_CALLING_CONV GetErrorStringA](#) ([eErr](#) [errCode](#))
- [EXTERN_C CCAUXDLL_API wchar_t const *CCAUXDLL_CALLING_CONV GetErrorStringW](#) ([eErr](#) [errCode](#))
- [EXTERN_C CCAUXDLL_API char const *CCAUXDLL_CALLING_CONV GetHwErrorStatusStringA](#) (unsigned short [errCode](#))
- [EXTERN_C CCAUXDLL_API wchar_t const *CCAUXDLL_CALLING_CONV GetHwErrorStatusStringW](#) (unsigned short [errCode](#))

- EXTERN_C CCAUXDLL_API char const *CCAUXDLL_CALLING_CONV [GetStartupReasonStringA](#) (unsigned short code)
- EXTERN_C CCAUXDLL_API wchar_t const *CCAUXDLL_CALLING_CONV [GetStartupReasonStringW](#) (unsigned short code)

Variables

- const unsigned char [Video1Conf](#) = (1 << 0)
- const unsigned char [Video2Conf](#) = (1 << 1)
- const unsigned char [Video3Conf](#) = (1 << 2)
- const unsigned char [Video4Conf](#) = (1 << 3)
- const unsigned char [DigitalIn_1](#) = (1 << 0)
- const unsigned char [DigitalIn_2](#) = (1 << 1)
- const unsigned char [DigitalIn_3](#) = (1 << 2)
- const unsigned char [DigitalIn_4](#) = (1 << 3)

4.1.1 Typedef Documentation

4.1.1.1 typedef struct version_info CrossControl::VersionType

4.1.2 Enumeration Type Documentation

4.1.2.1 enum CrossControl::ButtonPowerTransitionStatus

Current status for front panel button and on/off signal. If any of them generate a suspend or shutdown event, it can also be read, briefly. When the button/signal is released, typically BPTS_Suspend or BPTS_ShutDown follows.

Enumerator:

BPTS_No_Change No change

BPTS_ShutDown A shutdown has been initiated since the front panel button has been pressed longer than the set FrontBtnShutDownTrigTime

BPTS_Suspend Suspend mode has been initiated since the front panel button has been pressed (shortly) and suspend mode is enabled

BPTS_Restart Not currently in use

BPTS_BtnPressed The front panel button is currently pressed. It has not been released and it has not yet been held longer than FrontBtnShutDownTrigTime.

BPTS_BtnPressedLong The front panel button is currently pressed. It has not been released and it has been held longer than FrontBtnShutDownTrigTime.

BPTS_SignalOff The external on/off signal is low, but not yet long enough for the ExtOnOffSigSuspTrigTime.

4.1.2.2 enum CrossControl::CalibrationConfigParam

Touch screen calibration parameters

Enumerator:

CONFIG_CALIBRATION_WITH

CONFIG_CALIBRATION_MEASUREMENTS Accepted error value when calibrating.

CONFIG_5P_CALIBRATION_POINT_BORDER Number of measurements to accept a calibration point.

CONFIG_13P_CALIBRATION_POINT_BORDER The number of pixels from the border where the 5 point calibration points should be located.

CONFIG_13P_CALIBRATION_TRANSITION_MIN The number of pixels from the border where the 13 point calibration points should be located.

CONFIG_13P_CALIBRATION_TRANSITION_MAX Min defines the transition area in number of pixels, where the two different calibrations are used.

4.1.2.3 enum CrossControl::CalibrationModeSettings

Touch screen calibration modes

Enumerator:

MODE_UNKNOWN

MODE_NORMAL Unknown mode.

MODE_CALIBRATION_5P Normal operation mode.

MODE_CALIBRATION_9P Calibration with 5 points mode.

MODE_CALIBRATION_13P Calibration with 9 points mode.

4.1.2.4 enum CrossControl::CanFrameType

Can frame type settings

Enumerator:

FrameStandard

FrameExtended

FrameStandardExtended

4.1.2.5 enum CrossControl::CCAuxColor

Enumeration of standard colors

Enumerator:

RED
GREEN RGB 0xF, 0x0, 0x0
BLUE RGB 0x0, 0xF, 0x0
CYAN RGB 0x0, 0x0, 0xF
MAGENTA RGB 0x0, 0xF, 0xF
YELLOW RGB 0xF, 0x0, 0xF
UNDEFINED_COLOR RGB 0xF, 0xF, 0x0
Returns if color is not a standard color

4.1.2.6 enum CrossControl::CCStatus

Enable/disable enumeration

Enumerator:

Disabled
Enabled The setting is disabled or turned off

4.1.2.7 enum CrossControl::DeInterlaceMode

Enumerator:

DeInterlace_Even
DeInterlace_Odd Use only even rows from the interlaced input stream
DeInterlace_BOB Use only odd rows from the interlaced input stream

4.1.2.8 enum CrossControl::eErr

Error code enumeration

Enumerator:

ERR_SUCCESS
ERR_OPEN_FAILED Success
ERR_NOT_SUPPORTED Open failed
ERR_UNKNOWN_FEATURE Not supported
ERR_DATATYPE_MISMATCH Unknown feature
ERR_CODE_NOT_EXIST Datatype mismatch
ERR_BUFFER_SIZE Code doesn't exist
ERR_IOCTL_FAILED Buffer size error
ERR_INVALID_DATA IoCtrl operation failed

ERR_INVALID_PARAMETER Invalid data
ERR_CREATE_THREAD Invalid parameter
ERR_IN_PROGRESS Failed to create thread
ERR_CHECKSUM Operation in progress
ERR_INIT_FAILED Checksum error
ERR_VERIFY_FAILED Initialization failed
ERR_DEVICE_READ_DATA_FAILED Failed to verify
ERR_DEVICE_WRITE_DATA_FAILED Failed to read from device
ERR_COMMAND_FAILED Failed to write to device
ERR_EEPROM Command failed
ERR_JIDA_TEMP Error in EEPROM memory
ERR_AVERAGE_CALC_STARTED Failed to get JIDA temperature
ERR_NOT_RUNNING Calculation already started
ERR_I2C_EXPANDER_READ_FAILED Thread isn't running
ERR_I2C_EXPANDER_WRITE_FAILED I2C read failure
ERR_I2C_EXPANDER_INIT_FAILED I2C write failure
ERR_NEWER_SS_VERSION_REQUIRED I2C initialization failure
ERR_NEWER_FPGA_VERSION_REQUIRED SS version too old
ERR_NEWER_FRONT_VERSION_REQUIRED FPGA version too old
ERR_TELEMATICS_GPRS_NOT_AVAILABLE FRONT version too old
ERR_TELEMATICS_WLAN_NOT_AVAILABLE GPRS module not available

ERR_TELEMATICS_BT_NOT_AVAILABLE WLAN module not available
ERR_TELEMATICS_GPS_NOT_AVAILABLE Bluetooth module not available

ERR_MEM_ALLOC_FAIL GPS module not available

4.1.2.9 enum CrossControl::hwErrorStatusCodes

HW Error code enumeration. The codes that can be returned from getHwErrorStatus.

Enumerator:

errCodeNoErr
errCodeFPGACONFReadErr
errCodeFPGACONFUnexpVal
errCodeCBRESETReadErr
errCodeSUS3ReadErr
errCodeSUS4ReadErr

errCodeSUS5ReadErr
errCodePG5VSTBYReadErr
errCodePG5VSTBYUnexpVal
errCodeCANPWROKReadErr
errCodeVIDPWROKReadErr
errCodeLVDSBLENReadErr
errCodeLVDSVDDENReadErr
errCodeEXTCTRLONReadErr
errCodeFPBTNONReadErr
errCode24VReadErr
errCode24VOutOfLimits
errCode24VINReadErr
errCode24VINOutOfLimits
errCode12VReadErr
errCode12VOutOfLimits
errCode12VVIDEORReadErr
errCode12VVIDEOOutOfLimits
errCode5VSTBYReadErr
errCode5VSTBYOutOfLimits
errCode5VReadErr
errCode5VOutOfLimits
errCode3V3ReadErr
errCode3V3OutOfLimits
errCodeTFTVOLReadErr
errCodeTFTVLOOutOfLimits
errCode1V9ReadErr
errCode1V9OutOfLimits
errCode1V8ReadErr
errCode1V8OutOfLimits
errCode1V5ReadErr
errCode1V5OutOfLimits
errCode1V2ReadErr
errCode1V2OutOfLimits
errCode1V05ReadErr
errCode1V05OutOfLimits
errCode1V0ReadErr
errCode1V0OutOfLimits
errCode0V9ReadErr

errCodeOV9OutOfLimits
errCodeI2CTEMPReadErr
errCodeI2CTEMPOutOfLimits
errCodeSTM32TEMPReadErr
errCodeSTM32TEMPOutOfLimits
errCodeBLTYPEUnexpEEPROMVal
errCodeFPBTNUnexpEEPROMVal
errCodeEXTCTRLUnexpEEPROMVal
errCodeLowRange24VUnexpEEPROMVal
errCodeSuspToRAMUnexpEEPROMVal
errCodeCANPWRUnexpEEPROMVal
errCodeVID1PWRUnexpEEPROMVal
errCodeVID2PWRUnexpEEPROMVal
errCodeVID3PWRUnexpEEPROMVal
errCodeVID4PWRUnexpEEPROMVal
errCodeEXTFANUnexpEEPROMVal
errCodeLEDUnexpEEPROMVal
errCodeUnitTypeUnexpEEPROMVal
errCodeBLTYPEReadErrEEPROM
errCodeFPBTNReadErrEEPROM
errCodeEXTCTRLReadErrEEPROM
errCodeMaxSuspTimeReadErrEEPROM
errCodeLowRange24VReadErrEEPROM
errCodeSuspToRAMReadErrEEPROM
errCodeCANPWRReadErrEEPROM
errCodeVID1PWRReadErrEEPROM
errCodeVID2PWRReadErrEEPROM
errCodeVID3PWRReadErrEEPROM
errCodeVID4PWRReadErrEEPROM
errCodeEXTFANReadErrEEPROM
errCodeLEDReadErrEEPROM
errCodeUnitTypeReadErrEEPROM
errCodeRCCInit
errCodeDriverInit
errCodeSetSUPPLYRESET
errCodeRelSUPPLYRESET
errCodeSetSYSRESET
errCodeRelSYSRESET

errCodeSetPWRBTN
errCodeRelPWRBTN
errCodeOnBL
errCodeOffBL
errCodeEXTFANOn
errCodeEXTFANOff
errCodePWRENOn
errCodePWRENOff
errCodeMPPWRENOn
errCodeMPPWRENOff
errCodeCANPWRENOn
errCodeCANPWRENOff
errCodeVID1PWRENOn
errCodeVID1PWRENOff
errCodeVID2PWRENOn
errCodeVID2PWRENOff
errCodeVID3PWRENOn
errCodeVID3PWRENOff
errCodeVID4PWRENOn
errCodeVID4PWRENOff
errCodeHEATACTOn
errCodeHEATACTOff
errCodeSetLEDCol
errCodeSetLEDFreq
errCodeManageLED
errCodeManageCANPwr
errCodeManageMPPwr
errCodeManageVidPwr
errCodeManagePowSup
errCodeManageReset
errCodeSSState
errCodeVarWrapAround
errCodeFPBTNUexpVal
errCodeEXTCTRLUnexpVal
errCodeMAINPWROKReadErr
errCodeFRONTSPAREReadErr
errCodeTIMERReadErr
errCodeManageDiagnostics

errCodeFPBTNTimOutReadErrEEPROM
errCodeEXTCTRLTimOutReadErrEEPROM
errCodeFPBTNAndExtCtrlDisabled
errCodeSWVerReadErr
errCodeSWVerWriteErr
errCodeManageActDeAct
errCodeTickTimeOutTimer
errCodeOperateModeStateError
errCodeHeatingTempReadErrEEPROM
errCodeMPFailedStart
errCodeReadErrEEPROM
errCodeTimeOutWaitingForVoltages
errCodeMAX

4.1.2.10 enum CrossControl::JidaSensorType

Jida temperature sensor types

Enumerator:

TEMP_CPU
TEMP_BOX
TEMP_ENV
TEMP_BOARD
TEMP_BACKPLANE
TEMP_CHIPSETS
TEMP_VIDEO
TEMP_OTHER

4.1.2.11 enum CrossControl::LightSensorOperationRange

Light sensor operation ranges.

Enumerator:

RangeStandard
RangeExtended Light sensor operation range standard

4.1.2.12 enum CrossControl::LightSensorSamplingMode

Light sensor sampling modes.

Enumerator:

SamplingModeStandard

SamplingModeExtended Standard sampling mode.

SamplingModeAuto Extended sampling mode.

Auto switch between standard and extended sampling mode depending on saturation.

4.1.2.13 enum CrossControl::PowerAction

Button and on/off signal actions.

Enumerator:

NoAction No action taken

ActionSuspend The system enters suspend mode

ActionShutDown The system shuts down

4.1.2.14 enum CrossControl::shutDownReasonCodes

The shutdown codes returned by getShutDownReason.

Enumerator:

shutdownReasonCodeNoError

4.1.2.15 enum CrossControl::startupReasonCodes

The restart codes returned by getStartupReason.

Enumerator:

startupReasonCodeUndefined

startupReasonCodeButtonPress Unknown startup reason.

startupReasonCodeExtCtrl The system was started by front panel button press

startupReasonCodeMPRestart The system was started by the external control signal

startupReasonCodePowerOnStartup The system was restarted by OS request

4.1.2.16 enum CrossControl::TouchScreenModeSettings

Touch screen USB profile settings

Enumerator:

MOUSE_NEXT_BOOT

TOUCH_NEXT_BOOT Set the touch USB profile to mouse profile. Active upon the next boot.

MOUSE_NOW Set the touch USB profile to touch profile. Active upon the next boot.

TOUCH_NOW Immediately set the touch USB profile to mouse profile.

4.1.2.17 enum CrossControl::TriggerConf

Trigger configuration enumeration. Valid settings for enabling of front button and external on/off signal.

Enumerator:

Front_Button_Enabled Front button is enabled for startup and wake-up

OnOff_Signal_Enabled The external on/off signal is enabled for startup and wake-up

Both_Button_And_Signal_Enabled Both of the above are enabled

4.1.2.18 enum CrossControl::TSAdvancedSettingsParameter

Touch screen advanced settings parameters

Enumerator:

TS_RIGHT_CLICK_TIME Right click time in ms, for the mouse profile only.

TS_LOW_LEVEL Lowest A/D value required for registering a touch event. - Front uc 0.5.3.1 hand the default value of 3300, newer versions: 3400.

TS_UNTOUCHLEVEL A/D value where the screen is considered to be untouched.

TS_DEBOUNCE_TIME Debounce time is the time after first detected touch event during which no measurements are being taken. This is used to avoid faulty measurements that frequently happens right after the actual touch event. Front uc 0.5.3.1 hand the default value of 3ms, newer versions: 24ms.

TS_DEBOUNCE_TIMEOUT_TIME After debounce, an event will be ignored if after this time there are no valid measurements above *TS_LOW_LEVEL*. This time must be larger than *TS_DEBOUNCE_TIME*. Front uc 0.5.3.1 hand the default value of 12ms, newer versions: 36ms.

TS_DOUBLECLICK_MAX_CLICK_TIME Parameter used for improving double click accuracy. A touch event this long or shorter is considered to be one of the clicks in a double click.

TS_DOUBLE_CLICK_TIME Parameter used for improving double click accuracy. Time allowed between double clicks. Used for double click improvement.

TS_MAX_RIGHTCLICK_DISTANCE Maximum distance allowed to move pointer and still consider the event a right click.

TS_USE_DEJITTER The dejitter function enables smoother pointer movement. Set to non-zero to enable the function or zero to disable it.

TS_CALIBTATION_WIDTH Accepted difference in measurement during calibration of a point.

TS_CALIBRATION_MEASUREMENTS Number of measurements needed to accept a calibration point.

TS_RESTORE_DEFAULT_SETTINGS Set to non-zero to restore all the above settings to their defaults. This parameter cannot be read and setting it to zero has no effect.

4.1.2.19 enum CrossControl::UpgradeAction

Upgrade Action enumeration

Enumerator:

UPGRADE_INIT

UPGRADE_PREP_COM Initiating, checking for compatibility etc

UPGRADE_READING_FILE Preparing communication

UPGRADE_CONVERTING_FILE Opening and reading the supplied file

UPGRADE_FLASHING Converting the mcs format to binary format

UPGRADE_VERIFYING Flashing the file

UPGRADE_COMPLETE Verifying the programmed image

UPGRADE_COMPLETE_WITH_ERRORS Upgrade was finished

Upgrade finished prematurely, see errorCode for the reason of failure

4.1.2.20 enum CrossControl::VideoChannel

The available analog video channels

Enumerator:

Analog_Channel_1

Analog_Channel_2

Analog_Channel_3

Analog_Channel_4

4.1.2.21 enum CrossControl::videoStandard

Enumerator:

STD_M_J_NTSC
STD_B_D_G_H_I_N_PAL (M,J) NTSC ITU-R BT6.01
STD_M_PAL (B, D, G, H, I, N) PAL ITU-R BT6.01
STD_PAL (M) PAL ITU-R BT6.01
STD_NTSC PAL-Nc ITU-R BT6.01
STD_SECAM NTSC 4.43 ITU-R BT6.01

4.1.2.22 enum CrossControl::VoltageEnum

Voltage type enumeration

Enumerator:

VOLTAGE_24VIN
VOLTAGE_24V < 24VIN
VOLTAGE_12V < 24V
VOLTAGE_12VID < 12V
VOLTAGE_5V < 12VID
VOLTAGE_3V3 < 5V
VOLTAGE_VTFT < 3.3V
VOLTAGE_5VSTB < VTFT
VOLTAGE_1V9 < 5VSTB
VOLTAGE_1V8 < 1.9V
VOLTAGE_1V5 < 1.8V
VOLTAGE_1V2 < 1.5V
VOLTAGE_1V05 < 1.2V
VOLTAGE_1V0 < 1.05V
VOLTAGE_0V9 < 1.0V
VOLTAGE_VREF_INT < 0.9V
< SS internal VRef

4.1.3 Function Documentation

4.1.3.1 EXTERN_C CCAUXDLL_API char const* CCAUXDLL_CALLING_CONV
CrossControl::GetErrorStringA (eErr *errCode*)

CCAux Error handling

Get a string description of an error code.

Parameters

<i>errCode</i>	An error code for which to get a string description.
----------------	--

Returns

String description of an error code.

**4.1.3.2 EXTERN_C CCAUXDLL_API wchar_t const* CCAUXDLL_CALLING_CONV
CrossControl::GetErrorStringW (eErr *errCode*)**

Get a string description of an error code.

Parameters

<i>errCode</i>	An error code for which to get a string description.
----------------	--

Returns

String description of an error code.

**4.1.3.3 EXTERN_C CCAUXDLL_API char const* CCAUXDLL_CALLING_CONV
CrossControl::GetHwErrorStatusStringA (unsigned short *errCode*)**

String access functions for codes used in the diagnostic functions

Get a string description of an error code returned from getHwErrorStatus.

Parameters

<i>errCode</i>	An error code for which to get a string description.
----------------	--

Returns

String description of an error code.

**4.1.3.4 EXTERN_C CCAUXDLL_API wchar_t const* CCAUXDLL_CALLING_CONV
CrossControl::GetHwErrorStatusStringW (unsigned short *errCode*)**

Get a string description of an error code returned from getHwErrorStatus.

Parameters

<i>errCode</i>	An error code for which to get a string description.
----------------	--

Returns

String description of an error code.

4.1.3.5 **EXTERN_C CCAUXDLL_API char const* CCAUXDLL_CALLING_CONV**
CrossControl::GetStartupReasonStringA (unsigned short *code*)

Get a string description of a startup reason code returned from getStartupReason.

Parameters

<i>code</i>	A code for which to get a string description.
-------------	---

Returns

String description of a code.

4.1.3.6 **EXTERN_C CCAUXDLL_API wchar_t const* CCAUXDLL_CALLING_CONV**
CrossControl::GetStartupReasonStringW (unsigned short *code*)

Get a string description of a startup reason code returned from getStartupReason.

Parameters

<i>code</i>	A code for which to get a string description.
-------------	---

Returns

String description of a code.

4.1.4 Variable Documentation

4.1.4.1 **const unsigned char CrossControl::DigitalIn_1 = (1 << 0)**

Bit defines for getDigIO

4.1.4.2 **const unsigned char CrossControl::DigitalIn_2 = (1 << 1)**

4.1.4.3 **const unsigned char CrossControl::DigitalIn_3 = (1 << 2)**

4.1.4.4 **const unsigned char CrossControl::DigitalIn_4 = (1 << 3)**

4.1.4.5 **const unsigned char CrossControl::Video1Conf = (1 << 0)**

Bit defines for getVideoStartupPowerConfig and setVideoStartupPowerConfig

4.1.4.6 const unsigned char `CrossControl::Video2Conf` = (1 << 1)

[Video](#) channel 1 config

4.1.4.7 const unsigned char `CrossControl::Video3Conf` = (1 << 2)

[Video](#) channel 2 config

4.1.4.8 const unsigned char `CrossControl::Video4Conf` = (1 << 3)

[Video](#) channel 3 config

Chapter 5

Class Documentation

5.1 CrossControl::About Struct Reference

```
#include <About.h>
```

Public Member Functions

- virtual [eErr getMainPCBSerial](#) (char *buff, int len)=0
- virtual [eErr getUnitSerial](#) (char *buff, int len)=0
- virtual [eErr getMainPCBArt](#) (char *buff, int length)=0
- virtual [eErr getMainManufacturingDate](#) (char *buff, int len)=0
- virtual [eErr getMainHWversion](#) (char *buff, int len)=0
- virtual [eErr getMainProdRev](#) (char *buff, int len)=0
- virtual [eErr getMainProdArtNr](#) (char *buff, int length)=0
- virtual [eErr getNrOfETHConnections](#) (unsigned char *NrOfConnections)=0
- virtual [eErr getNrOfCANConnections](#) (unsigned char *NrOfConnections)=0
- virtual [eErr getNrOfVideoConnections](#) (unsigned char *NrOfConnections)=0
- virtual [eErr getNrOfUSBConnections](#) (unsigned char *NrOfConnections)=0
- virtual [eErr getNrOfSerialConnections](#) (unsigned char *NrOfConnections)=0
- virtual [eErr getAddOnPCBSerial](#) (char *buff, int length)=0
- virtual [eErr getAddOnPCBArt](#) (char *buff, int length)=0
- virtual [eErr getAddOnManufacturingDate](#) (char *buff, int length)=0
- virtual [eErr getAddOnHWversion](#) (char *buff, int length)=0
- virtual [eErr getIsWLANMounted](#) (bool *mounted)=0
- virtual [eErr getIsGPSMounted](#) (bool *mounted)=0
- virtual [eErr getIsGPRSMounted](#) (bool *mounted)=0
- virtual [eErr getIsBTMounted](#) (bool *mounted)=0
- virtual void [Release](#) ()=0
- virtual [eErr getNrOfDigIOConnections](#) (unsigned char *NrOfConnections)=0
- virtual [eErr getIsDisplayAvailable](#) (bool *available)=0
- virtual [eErr getIsTouchScreenAvailable](#) (bool *available)=0

5.1.1 Detailed Description

Get information about the CCpilot XM computer.

Use the globally defined function [GetAbout\(\)](#) to get a handle to the [About](#) struct. Use the method [About::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/about_example.cpp -lcc-aux -pthread -ldl */
#include <About.h>
#include <assert.h>
#include <iostream>

using namespace std;

void list_about_information(ABOUTHANDLE pAbout)
{
    if(!pAbout)
        return;

    size_t const buffer_len = 256;
    char* buffer = new (nothrow) char[buffer_len];

    if(!buffer)
        return;

    CrossControl::eErr err;

    err = pAbout->getMainPCBSerial (buffer, buffer_len);
    if (CrossControl::ERR_SUCCESS == err)
        cout << "Main PCB serial: " << buffer << endl;

    err = pAbout->getMainPCBArt (buffer, buffer_len);
    if (CrossControl::ERR_SUCCESS == err)
        cout << "Main PCB article number: " << buffer << endl;

    err = pAbout->getUnitSerial (buffer, buffer_len);
    if (CrossControl::ERR_SUCCESS == err)
        cout << "Unit serial: " << buffer << endl;

    err = pAbout->getMainManufacturingDate (buffer, buffer_len);
    if (CrossControl::ERR_SUCCESS == err)
        cout << "Manufacturing date: " << buffer << endl;

    err = pAbout->getMainHWversion (buffer, buffer_len);
    if (CrossControl::ERR_SUCCESS == err)
        cout << "Main hardware version: " << buffer << endl;

    err = pAbout->getMainProdRev (buffer, buffer_len);
    if (CrossControl::ERR_SUCCESS == err)
        cout << "Main product revision: " << buffer << endl;

    err = pAbout->getMainProdArtNr (buffer, buffer_len);
    if (CrossControl::ERR_SUCCESS == err)
        cout << "Main product article number: " << buffer << endl;

    err = pAbout->getAddOnPCBSerial (buffer, buffer_len);
    if (CrossControl::ERR_SUCCESS == err)
        cout << "Add on PCB serial number: " << buffer << endl;

    err = pAbout->getAddOnPCBArt (buffer, buffer_len);
```

```
if (CrossControl::ERR_SUCCESS == err)
    cout << "Add on PCB article number: " << buffer << endl;

err = pAbout->getAddOnManufacturingDate (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Add on manufacturing date: " << buffer << endl;

err = pAbout->getAddOnHWversion (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Add on hardware version: " << buffer << endl;

unsigned char nrOfEthConnections;
err = pAbout->getNrOfETHConnections (&nrOfEthConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of ethernet connections: " << nrOfEthConnections << endl;

unsigned char nrOfCANConnections;
err = pAbout->getNrOfCANConnections (&nrOfCANConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of CAN connections: " << nrOfCANConnections << endl;

unsigned char nrOfVideoConnections;
err = pAbout->getNrOfVideoConnections (&nrOfVideoConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of video connections: " << nrOfVideoConnections << endl;

unsigned char nrOfUSBConnections;
err = pAbout->getNrOfUSBConnections (&nrOfUSBConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of USB connections: " << nrOfUSBConnections << endl;

unsigned char nrOfSerialConnections;
err = pAbout->getNrOfSerialConnections (&nrOfSerialConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of serial connections: " << nrOfSerialConnections << endl;

unsigned char nrOfDigIOConnections;
err = pAbout->getNrOfDigIOConnections (&nrOfDigIOConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of digital I/O connections: " << nrOfDigIOConnections << endl;

bool displayAvailable;
err = pAbout->getIsDisplayAvailable (&displayAvailable);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Display available: " << (displayAvailable ? "YES" : "NO") << endl;

bool touchScreenAvailable;
err = pAbout->getIsTouchScreenAvailable (&touchScreenAvailable);
if (CrossControl::ERR_SUCCESS == err)
    cout << "TouchScreen available: " << (touchScreenAvailable ? "YES" : "NO")
        << endl;

bool isWLANMounted;
err = pAbout->getIsWLANMounted (&isWLANMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "WLAN mounted: " << (isWLANMounted ? "YES" : "NO") << endl;

bool isGPSPmounted;
err = pAbout->getIsGPSPmounted (&isGPSPmounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "GPS mounted: " << (isGPSPmounted ? "YES" : "NO") << endl;
```

```

bool isGPRSMounted;
err = pAbout->getIsGPRSMounted (&isGPRSMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "GPRS mounted: " << (isGPRSMounted ? "YES" : "NO") << endl;

bool isBTMounted;
err = pAbout->getIsBTMounted (&isBTMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "BT mounted: " << (isBTMounted ? "YES" : "NO") << endl;

delete[] buffer;
}

int main(void)
{
    ABOUTHANDLE pAbout = ::GetAbout();
    assert(pAbout);

    list_about_information(pAbout);

    pAbout->Release();
}

```

5.1.2 Member Function Documentation

5.1.2.1 virtual eErr CrossControl::About::getAddOnHWversion (char * *buff*, int *length*) [pure virtual]

Get Add on hardware version.

Parameters

<i>buff</i>	Text output buffer.
<i>length</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pAbout->getAddOnHWversion (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Add on hardware version: " << buffer << endl;

```

5.1.2.2 virtual eErr CrossControl::About::getAddOnManufacturingDate (char * *buff*, int *length*) [pure virtual]

Get Add on manufacturing date.

Parameters

<i>buff</i>	Text output buffer.
<i>length</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getAddOnManufacturingDate (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Add on manufacturing date: " << buffer << endl;
```

5.1.2.3 virtual eErr CrossControl::About::getAddOnPCBArt (char * buff, int length) [pure virtual]

Get Add on PCB article number.

Parameters

<i>buff</i>	Text output buffer.
<i>length</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getAddOnPCBArt (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Add on PCB article number: " << buffer << endl;
```

5.1.2.4 virtual eErr CrossControl::About::getAddOnPCBSerial (char * buff, int length) [pure virtual]

Get Add on PCB serial number.

Parameters

<i>buff</i>	Text output buffer.
<i>length</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getAddOnPCBSerial (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Add on PCB serial number: " << buffer << endl;
```

5.1.2.5 virtual eErr CrossControl::About::getIsBTMounted (bool * *mounted*) [pure virtual]

Get BlueTooth module mounting status.

Parameters

<i>mounted</i>	Is module mounted?
----------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
bool isBTMounted;
err = pAbout->getIsBTMounted (&isBTMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "BT mounted: " << (isBTMounted ? "YES" : "NO") << endl;
```

5.1.2.6 virtual eErr CrossControl::About::getIsDisplayAvailable (bool * *available*) [pure virtual]

Get Display module status. (Some product variants does not have a display)

Parameters

<i>available</i>	Is display available?
------------------	-----------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
bool displayAvailable;
err = pAbout->getIsDisplayAvailable (&displayAvailable);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Display available: " << (displayAvailable ? "YES" : "NO") << endl;
```

5.1.2.7 virtual eErr CrossControl::About::getIsGPRSMounted (bool * *mounted*) [pure virtual]

Get GPRS module mounting status.

Parameters

<i>mounted</i>	Is module mounted?
----------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
bool isGPRSMounted;
err = pAbout->getIsGPRSMounted (&isGPRSMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "GPRS mounted: " << (isGPRSMounted ? "YES" : "NO") << endl;
```

5.1.2.8 virtual eErr CrossControl::About::getIsGPSMounted (bool * *mounted*) [pure virtual]

Get GPS module mounting status.

Parameters

<i>mounted</i>	Is module mounted?
----------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
bool isGPSMounted;
err = pAbout->getIsGPSMounted (&isGPSMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "GPS mounted: " << (isGPSMounted ? "YES" : "NO") << endl;
```

5.1.2.9 virtual eErr CrossControl::About::getIsTouchScreenAvailable (bool * *available*) [pure virtual]

Get Display [TouchScreen](#) status.

Parameters

<i>available</i>	Is TouchScreen available?
------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
bool touchScreenAvailable;
err = pAbout->getIsTouchScreenAvailable (&touchScreenAvailable);
if (CrossControl::ERR_SUCCESS == err)
    cout << "TouchScreen available: " << (touchScreenAvailable ? "YES" : "NO")
        << endl;
```

5.1.2.10 virtual eErr CrossControl::About::getIsWLANMounted (bool * *mounted*) [pure virtual]

Get WLAN module mounting status.

Parameters

<i>mounted</i>	Is module mounted?
----------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
bool isWLANMounted;
err = pAbout->getIsWLANMounted (&isWLANMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "WLAN mounted: " << (isWLANMounted ? "YES" : "NO") << endl;
```

5.1.2.11 virtual eErr CrossControl::About::getMainHWversion (char * *buff*, int *len*) [pure virtual]

Get main hardware version (PCB revision).

Parameters

<i>buff</i>	Text output buffer.
<i>len</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getMainHWversion (buffer, buffer_len);
```

```
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main hardware version: " << buffer << endl;
```

5.1.2.12 virtual eErr CrossControl::About::getMainManufacturingDate (char * *buff*, int *len*) [pure virtual]

Get main manufacturing date.

Parameters

<i>buff</i>	Text output buffer.
<i>len</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getMainManufacturingDate (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Manufacturing date: " << buffer << endl;
```

5.1.2.13 virtual eErr CrossControl::About::getMainPCBArt (char * *buff*, int *length*) [pure virtual]

Get main PCB article number.

Parameters

<i>buff</i>	Text output buffer.
<i>length</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getMainPCBArt (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main PCB article number: " << buffer << endl;
```

5.1.2.14 virtual eErr CrossControl::About::getMainPCBSerial (char * *buff*, int *len*) [pure virtual]

Get main PCB serial number.

Parameters

<i>buff</i>	Text output buffer.
<i>len</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getMainPCBSerial (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main PCB serial: " << buffer << endl;
```

5.1.2.15 virtual eErr CrossControl::About::getMainProdArtNr (char * buff, int length) [pure virtual]

Get main product article number.

Parameters

<i>buff</i>	Text output buffer.
<i>length</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getMainProdArtNr (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main product article number: " << buffer << endl;
```

5.1.2.16 virtual eErr CrossControl::About::getMainProdRev (char * buff, int len) [pure virtual]

Get main product revision.

Parameters

<i>buff</i>	Text output buffer.
<i>len</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getMainProdRev (buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main product revision: " << buffer << endl;
```

5.1.2.17 virtual eErr CrossControl::About::getNrOfCANConnections (unsigned char * *NrOfConnections*) [pure virtual]

Get number of CAN connections present.

Parameters

<i>NrOf-Connections</i>	Returns the number of connections.
-------------------------	------------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char nrOfCANConnections;
err = pAbout->getNrOfCANConnections (&nrOfCANConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of CAN connections: " << nrOfCANConnections << endl;
```

5.1.2.18 virtual eErr CrossControl::About::getNrOfDigIOConnections (unsigned char * *NrOfConnections*) [pure virtual]

Get number of digital I/O connections present.

Parameters

<i>NrOf-Connections</i>	Returns the number of input or input/output connections.
-------------------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char nrOfDigIOConnections;
err = pAbout->getNrOfDigIOConnections (&nrOfDigIOConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of digital I/O connections: " << nrOfDigIOConnections << endl;
```

5.1.2.19 `virtual eErr CrossControl::About::getNrOfETHConnections (unsigned char * NrOfConnections) [pure virtual]`

Get number of ethernet connections present.

Parameters

<i>NrOf-Connections</i>	Returns the number of connections.
-------------------------	------------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char nrOfEthConnections;
err = pAbout->getNrOfETHConnections (&nrOfEthConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of ethernet connections: " << nrOfEthConnections << endl;
```

5.1.2.20 `virtual eErr CrossControl::About::getNrOfSerialConnections (unsigned char * NrOfConnections) [pure virtual]`

Get number of serial port (RS232) connections present.

Parameters

<i>NrOf-Connections</i>	Returns the number of connections.
-------------------------	------------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char nrOfSerialConnections;
err = pAbout->getNrOfSerialConnections (&nrOfSerialConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of serial connections: " << nrOfSerialConnections << endl;
```

5.1.2.21 `virtual eErr CrossControl::About::getNrOfUSBConnections (unsigned char * NrOfConnections) [pure virtual]`

Get number of USB connections present.

Parameters

<i>NrOf-Connections</i>	Returns the number of connections.
-------------------------	------------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char nrOfUSBConnections;
err = pAbout->getNrOfUSBConnections (&nrOfUSBConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of USB connections: " << nrOfUSBConnections << endl;
```

5.1.2.22 virtual eErr CrossControl::About::getNrOfVideoConnections (unsigned char * *NrOfConnections*) [pure virtual]

Get number of [Video](#) connections present.

Parameters

<i>NrOf-Connections</i>	Returns the number of connections.
-------------------------	------------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char nrOfVideoConnections;
err = pAbout->getNrOfVideoConnections (&nrOfVideoConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of video connections: " << nrOfVideoConnections << endl;
```

5.1.2.23 virtual eErr CrossControl::About::getUnitSerial (char * *buff*, int *len*) [pure virtual]

Get unit serial number.

Parameters

<i>buff</i>	Text output buffer.
<i>len</i>	Maximum length of the output buffer. If the actual length of the data is greater, an error will be returned.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAbout->getUnitSerial (buffer, buffer_len);
```

```
if (CrossControl::ERR_SUCCESS == err)
    cout << "Unit serial: " << buffer << endl;
```

5.1.2.24 virtual void CrossControl::About::Release () [pure virtual]

Delete the [About](#) object.

Returns

-

Example Usage:

```
ABOUTHANDLE pAbout = ::GetAbout();
assert(pAbout);

list_about_information(pAbout);

pAbout->Release();
```

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/About.h](#)

5.2 CrossControl::Adc Struct Reference

```
#include <Adc.h>
```

Public Member Functions

- virtual [CrossControl::eErr getVoltage](#) ([VoltageEnum](#) selection, double *value)=0
- virtual void [Release](#) ()=0

5.2.1 Detailed Description

Get current voltages from the built-in ADC

Use the globally defined function [GetAdc\(\)](#) to get a handle to the [Adc](#) struct. Use the method [Adc::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/adc_example.cpp -lcc-aux -pthread -ldl */
#include <Adc.h>
#include <assert.h>
#include <iomanip>
#include <iostream>

using namespace std;
```

```

void output_voltage(
    ADCHANDLE pAdc,
    char const* description,
    CrossControl::VoltageEnum selection)
{
    if(!pAdc)
        return;

    CrossControl::eErrr err;
    double voltage;

    err = pAdc->getVoltage (selection, &voltage);
    if (CrossControl::ERR_SUCCESS == err)
    {
        cout << left << setw(7) << description << ":" <<
            fixed << setprecision(2) << voltage << "V" << endl;
    }
}

int main(void)
{
    ADCHANDLE pAdc = ::GetAdc();
    assert(pAdc);

    output_voltage (pAdc, "24VIN", CrossControl::VOLTAGE_24VIN);
    output_voltage (pAdc, "24V", CrossControl::VOLTAGE_24V);
    output_voltage (pAdc, "12V", CrossControl::VOLTAGE_12V);
    output_voltage (pAdc, "12VID", CrossControl::VOLTAGE_12VID);
    output_voltage (pAdc, "5V", CrossControl::VOLTAGE_5V);
    output_voltage (pAdc, "3V3", CrossControl::VOLTAGE_3V3);
    output_voltage (pAdc, "VTFT", CrossControl::VOLTAGE_VTFT);
    output_voltage (pAdc, "5VSTB", CrossControl::VOLTAGE_5VSTB);
    output_voltage (pAdc, "1V9", CrossControl::VOLTAGE_1V9);
    output_voltage (pAdc, "1V8", CrossControl::VOLTAGE_1V8);
    output_voltage (pAdc, "1V5", CrossControl::VOLTAGE_1V5);
    output_voltage (pAdc, "1V2", CrossControl::VOLTAGE_1V2);
    output_voltage (pAdc, "1V05", CrossControl::VOLTAGE_1V05);
    output_voltage (pAdc, "1V0", CrossControl::VOLTAGE_1V0);
    output_voltage (pAdc, "0V9", CrossControl::VOLTAGE_0V9);

    pAdc->Release();
}

```

5.2.2 Member Function Documentation

5.2.2.1 virtual CrossControl::eErrr CrossControl::Adc::getVoltage (VoltageEnum selection, double * value) [pure virtual]

Read measured voltage.

Parameters

<i>selection</i>	The type of voltage to get.
<i>value</i>	Voltage value in Volt.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAdc->getVoltage (selection, &voltage);
if (CrossControl::ERR_SUCCESS == err)
{
    cout << left << setw(7) << description << ":" <<
        fixed << setprecision(2) << voltage << "V" << endl;
}
```

5.2.2.2 virtual void CrossControl::Adc::Release() [pure virtual]

Delete the ADC object.

Returns

-

Example Usage:

```
ADCHANDLE pAdc = ::GetAdc();
assert(pAdc);

output_voltage (pAdc, "24VIN", CrossControl::VOLTAGE_24VIN);
output_voltage (pAdc, "24V", CrossControl::VOLTAGE_24V);
output_voltage (pAdc, "12V", CrossControl::VOLTAGE_12V);
output_voltage (pAdc, "12VID", CrossControl::VOLTAGE_12VID);
output_voltage (pAdc, "5V", CrossControl::VOLTAGE_5V);
output_voltage (pAdc, "3V3", CrossControl::VOLTAGE_3V3);
output_voltage (pAdc, "VTFT", CrossControl::VOLTAGE_VTFT);
output_voltage (pAdc, "5VSTB", CrossControl::VOLTAGE_5VSTB);
output_voltage (pAdc, "1V9", CrossControl::VOLTAGE_1V9);
output_voltage (pAdc, "1V8", CrossControl::VOLTAGE_1V8);
output_voltage (pAdc, "1V5", CrossControl::VOLTAGE_1V5);
output_voltage (pAdc, "1V2", CrossControl::VOLTAGE_1V2);
output_voltage (pAdc, "1V05", CrossControl::VOLTAGE_1V05);
output_voltage (pAdc, "1V0", CrossControl::VOLTAGE_1V0);
output_voltage (pAdc, "0V9", CrossControl::VOLTAGE_0V9);

pAdc->Release();
```

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Adc.h](#)

5.3 CrossControl::AuxVersion Struct Reference

```
#include <AuxVersion.h>
```

Public Member Functions

- virtual [eErr getFPGAVersion](#) (unsigned char *major, unsigned char *minor, unsigned char *release, unsigned char *build)=0
- virtual [eErr getSSVersion](#) (unsigned char *major, unsigned char *minor, unsigned char *release, unsigned char *build)=0
- virtual [eErr getFrontVersion](#) (unsigned char *major, unsigned char *minor, unsigned char *release, unsigned char *build)=0
- virtual [eErr getCCAuxVersion](#) (unsigned char *major, unsigned char *minor, unsigned char *release, unsigned char *build)=0
- virtual [eErr getOSVersion](#) (unsigned char *major, unsigned char *minor, unsigned char *release, unsigned char *build)=0
- virtual [eErr getCCAuxDrvVersion](#) (unsigned char *major, unsigned char *minor, unsigned char *release, unsigned char *build)=0
- virtual void [Release](#) ()=0

5.3.1 Detailed Description

Get software versions for firmware and software

Use the globally defined function [GetAuxVersion\(\)](#) to get a handle to the [AuxVersion](#) * struct. Use the method [AuxVersion::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/auxversion_example.cpp -lcc-aux -pthread -ldl */
#include <assert.h>
#include <AuxVersion.h>
#include <iomanip>
#include <iostream>

using namespace std;

void output_versions(AUXVERSIONHANDLE pAuxVersion)
{
    if (!pAuxVersion)
        return;

    int const column_width = 32;
    unsigned char major, minor, release, build;
    CrossControl::eErr err;

    err = pAuxVersion->getFPGAVersion(
        &major,
        &minor,
        &release,
        &build);

    cout << setw(column_width) << "FPGA Version: ";
    if (CrossControl::ERR_SUCCESS != err)
        cout << (int) major << "." <<
            (int) minor << "." <<
            (int) release << "." <<
            (int) build << endl;
    else
        cout << "unknown" << endl;
}
```

```
err = pAuxVersion->getSSVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "System Supervisor Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;

err = pAuxVersion->getFrontVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "Front Micro Controller Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;

err = pAuxVersion->getCCAuxVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "CC Aux Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout <<
        (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;

err = pAuxVersion->getOSVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "Operating System Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;
```

```

err = pAuxVersion->getCCAuxDrvVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "CCAux Driver Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;
}

int main(void)
{
    AUXVERSIONHANDLE pAuxVersion = ::GetAuxVersion();
    assert (pAuxVersion);

    output_versions(pAuxVersion);

    pAuxVersion->Release();
}

```

5.3.2 Member Function Documentation

5.3.2.1 virtual eErr CrossControl::AuxVersion::getCCAuxDrvVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*)
 [pure virtual]

Get the [CrossControl](#) CCAux CCAuxDrv version. Can be used to check that the correct driver is loaded. The version should be the same as that of [getCCAuxVersion](#).

Parameters

<i>major</i>	Major version number
<i>minor</i>	Minor version number
<i>release</i>	Release version number
<i>build</i>	Build version number

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pAuxVersion->getCCAuxDrvVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "CCAux Driver Version: ";

```

```

if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;

```

5.3.2.2 virtual eErr CrossControl::AuxVersion::getCCAuxVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the [CrossControl](#) CCAux API version. CCAux includes: CCAuxDrv - Hardware driver. CCAuxService - Windows Service. ccauxd - Linux daemon. CCAuxDll - The dll implementing this API.

Parameters

<i>major</i>	Major version number
<i>minor</i>	Minor version number
<i>release</i>	Release version number
<i>build</i>	Build version number

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pAuxVersion->getCCAuxVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "CC Aux Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout <<
        (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;

```

5.3.2.3 virtual eErr CrossControl::AuxVersion::getFPGAVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the FPGA software version

Parameters

<i>major</i>	Major version number
<i>minor</i>	Minor version number
<i>release</i>	Release version number
<i>build</i>	Build version number

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAuxVersion->getFPGAVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "FPGA Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;
```

5.3.2.4 virtual eErr CrossControl::AuxVersion::getFrontVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the front microcontroller software version

Parameters

<i>major</i>	Major version number
<i>minor</i>	Minor version number
<i>release</i>	Release version number
<i>build</i>	Build version number

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAuxVersion->getFrontVersion(
    &major,
    &minor,
    &release,
    &build);
```

```

cout << setw(column_width) << "Front Micro Controller Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;

```

5.3.2.5 virtual eErr CrossControl::AuxVersion::getOSVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the [CrossControl](#) Operating System version.

Parameters

<i>major</i>	Major version number
<i>minor</i>	Minor version number
<i>release</i>	Release version number
<i>build</i>	Build version number

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pAuxVersion->getOSVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "Operating System Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;

```

5.3.2.6 virtual eErr CrossControl::AuxVersion::getSSVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the System Supervisor software version

Parameters

<i>major</i>	Major version number
<i>minor</i>	Minor version number
<i>release</i>	Release version number
<i>build</i>	Build version number

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pAuxVersion->getSSVersion(
    &major,
    &minor,
    &release,
    &build);

cout << setw(column_width) << "System Supervisor Version: ";
if (CrossControl::ERR_SUCCESS != err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;
```

5.3.2.7 virtual void CrossControl::AuxVersion::Release () [pure virtual]

Delete the [AuxVersion](#) object.

Returns

-

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/AuxVersion.h](#)

5.4 CrossControl::Backlight Struct Reference

```
#include <Backlight.h>
```

Public Member Functions

- virtual [eErr getIntensity](#) (unsigned char *intensity)=0
- virtual [eErr setIntensity](#) (unsigned char intensity)=0
- virtual [eErr getStatus](#) (unsigned char *status)=0

- virtual `eErr startAutomaticBL ()=0`
- virtual `eErr stopAutomaticBL ()=0`
- virtual `eErr getAutomaticBLStatus (unsigned char *status)=0`
- virtual `eErr setAutomaticBLParams (bool bSoftTransitions)=0`
- virtual `eErr getAutomaticBLParams (bool *bSoftTransitions, double *k)=0`
- virtual `eErr setAutomaticBLFilter (unsigned long averageWndSize, unsigned long rejectWndSize, unsigned long rejectDeltaInLux, LightSensorSamplingMode mode)=0`
- virtual `eErr getAutomaticBLFilter (unsigned long *averageWndSize, unsigned long *rejectWndSize, unsigned long *rejectDeltaInLux, LightSensorSamplingMode *mode)=0`
- virtual `eErr getLedDimming (CCStatus *status)=0`
- virtual `eErr setLedDimming (CCStatus status)=0`
- virtual void `Release ()=0`

5.4.1 Detailed Description

Backlight settings

Use the globally defined function `GetBacklight()` to get a handle to the `Backlight` struct. Use the method `Backlight::Release()` to return the handle.

Example Usage:

```

/* g++ -DLINUX examples/backlight_example.cpp -lcc-aux -pthread -ldl */
#include <Backlight.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>
#include <stdio.h>
using namespace std;
using namespace CrossControl;

void change_backlight (BACKLIGHTHANDLE pBacklight)
{
    if (!pBacklight)
        return;

    CrossControl::eErr err;
    unsigned char value;

    err = pBacklight->getStatus (&value);
    if (err == ERR_SUCCESS)
    {
        printf ("Backlight status: \nBL1:%s\nBL2:%s\nBL3:%s\nBL4:%s\n",
            (value & 0x01)? "OK" : "NOT OK or missing",
            (value & 0x02)? "OK" : "NOT OK or missing",
            (value & 0x04)? "OK" : "NOT OK or missing",
            (value & 0x08)? "OK" : "NOT OK or missing");
    }
    else
    {
        printf ("Error(%d) in function getStatus: %s\n", err, GetErrorStringA (err));
    }

    //Get current intensity

```

```

err = pBacklight->getIntensity(&value);

if(err == ERR_SUCCESS)
{
    printf("Current backlight intensity (0-255): %d\n", value);
}
else
{
    printf("Error(%d) in function getIntensity: %s\n", err, GetErrorStringA(err
));
}

if(value < 245)
    value = value + 10;
else
    value = value -10;

//Set current intensity
err = pBacklight->setIntensity(value);

if(err == ERR_SUCCESS)
{
    printf("Setting backlight intensity: %d\n", value);
}
else
{
    printf("Error(%d) in function setIntensity: %s\n", err, GetErrorStringA(err
));
}

}

int main(void)
{
    BACKLIGHTHANDLE pBacklight = ::GetBacklight();
    assert(pBacklight);

    change_backlight(pBacklight);

    pBacklight->Release();
}

```

5.4.2 Member Function Documentation

5.4.2.1 virtual eErr CrossControl::Backlight::getAutomaticBLFilter (unsigned long * *averageWndSize*, unsigned long * *rejectWndSize*, unsigned long * *rejectDeltaInLux*, LightSensorSamplingMode * *mode*) [pure virtual]

Get light sensor filter parameters for automatic backlight control.

Parameters

<i>average- WndSize</i>	The average window size in nr of samples.
<i>rejectWnd- Size</i>	The reject window size in nr of samples.

<i>rejectDelta- InLux</i>	The reject delta in lux.
<i>mode</i>	The configured sampling mode.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.4.2.2 virtual eErr CrossControl::Backlight::getAutomaticBLParams (bool * *bSoftTransitions*, double * *k*) [pure virtual]

Get parameters for automatic backlight control.

Parameters

<i>bSoft- Transitions</i>	Soft transitions used?
<i>k</i>	K value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.4.2.3 virtual eErr CrossControl::Backlight::getAutomaticBLStatus (unsigned char * *status*) [pure virtual]

Get status from automatic backlight control.

Parameters

<i>status</i>	1=running, 0=stopped.
---------------	-----------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.4.2.4 virtual eErr CrossControl::Backlight::getIntensity (unsigned char * *intensity*) [pure virtual]

Get backlight intensity. Note that the lowest value returned is 3.

Parameters

<i>intensity</i>	The current backlight intensity (3..255).
------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
//Get current intensity
err = pBacklight->getIntensity(&value);

if(err == ERR_SUCCESS)
{
    printf("Current backlight intensity (0-255): %d\n", value);
}
else
{
    printf("Error(%d) in function getIntensity: %s\n", err, GetErrorStringA(err));
}
```

5.4.2.5 virtual eErr CrossControl::Backlight::getLedDimming (CCStatus * *status*) [pure virtual]

Get the current setting for Led dimming. If enabled, the function automatically dims the LED according to the current backlight setting; Low backlight gives less bright LED. This works with manual backlight setting and automatic backlight, but only if the led is set to pure red, green or blue color. If another color is being used, this functionality must be implemented separately.

Parameters

<i>status</i>	Enabled/Disabled
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.4.2.6 virtual eErr CrossControl::Backlight::getStatus (unsigned char * *status*) [pure virtual]

Get backlight controller status.

Parameters

<i>status</i>	Backlight controller status. Bit 0: status controller 1. Bit 1: status controller 2. Bit 2: status controller 3. Bit 3: status controller 4. 1=normal, 0=fault.
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pBacklight->getStatus(&value);
if(err == ERR_SUCCESS)
{
    printf("Backlight status: \nBL1:%s\nBL2:%s\nBL3:%s\nBL4:%s\n",
        (value & 0x01)? "OK" : "NOT OK or missing",
        (value & 0x02)? "OK" : "NOT OK or missing",
        (value & 0x04)? "OK" : "NOT OK or missing",
        (value & 0x08)? "OK" : "NOT OK or missing");
}
else
{
    printf("Error(%d) in function getStatus: %s\n", err, GetErrorStringA(err));
}
```

5.4.2.7 virtual void CrossControl::Backlight::Release () [pure virtual]

Delete the backlight object.

Returns

-

Example Usage:

```
BACKLIGHTHANDLE pBacklight = ::GetBacklight();
assert(pBacklight);

change_backlight(pBacklight);

pBacklight->Release();
```

5.4.2.8 virtual eErr CrossControl::Backlight::setAutomaticBLFilter (unsigned long averageWndSize, unsigned long rejectWndSize, unsigned long rejectDeltaInLux, LightSensorSamplingMode mode) [pure virtual]

Set light sensor filter parameters for automatic backlight control.

Parameters

<i>average-WndSize</i>	The average window size in nr of samples.
<i>rejectWnd-Size</i>	The reject window size in nr of samples.
<i>rejectDelta-InLux</i>	The reject delta in lux.
<i>mode</i>	The configured sampling mode.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.4.2.9 virtual eErr CrossControl::Backlight::setAutomaticBLParams (bool *bSoftTransitions*) [pure virtual]

Set parameters for automatic backlight control.

Parameters

<i>bSoftTransitions</i>	Use soft transitions?
-------------------------	-----------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.4.2.10 virtual eErr CrossControl::Backlight::setIntensity (unsigned char *intensity*) [pure virtual]

Set backlight intensity. Note that setting a lower value than 3 actually sets the value 3. This is a hardware design limit.

Parameters

<i>intensity</i>	The backlight intensity to set (3..255).
------------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
//Set current intensity
err = pBacklight->setIntensity(value);

if(err == ERR_SUCCESS)
{
    printf("Setting backlight intensity: %d\n", value);
}
else
{
    printf("Error(%d) in function setIntensity: %s\n", err, GetErrorStringA(err));
}
```

5.4.2.11 `virtual eErr CrossControl::Backlight::setLedDimming (CCStatus status)`
 [pure virtual]

Enable/disable Led dimming. If enabled, the function automatically dims the LED according to the current backlight setting; Low backlight gives less bright LED. This works with manual backlight setting and automatic backlight, but only if the led is set to pure red, green or blue color. If another color is being used, this functionality must be implemented separately.

Parameters

<i>status</i>	Enabled/Disabled
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.4.2.12 `virtual eErr CrossControl::Backlight::startAutomaticBL ()` [pure virtual]

Start automatic backlight control. Note that reading the light sensor at the same time as running the automatic backlight control is not supported.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.4.2.13 `virtual eErr CrossControl::Backlight::stopAutomaticBL ()` [pure virtual]

Stop automatic backlight control.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Backlight.h](#)

5.5 CrossControl::Buzzer Struct Reference

```
#include <Buzzer.h>
```

Public Member Functions

- virtual [eErr getFrequency](#) (unsigned short *frequency)=0
- virtual [eErr getVolume](#) (unsigned short *volume)=0
- virtual [eErr getTrigger](#) (bool *trigger)=0
- virtual [eErr setFrequency](#) (unsigned short frequency)=0
- virtual [eErr setVolume](#) (unsigned short volume)=0
- virtual [eErr setTrigger](#) (bool trigger)=0
- virtual [eErr buzze](#) (int time, bool blocking)=0
- virtual void [Release](#) ()=0

5.5.1 Detailed Description

Buzzer settings

Use the globally defined function [GetBuzzer\(\)](#) to get a handle to the [Buzzer](#) struct. Use the method [Buzzer::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/buzzer_example.cpp -lcc-aux -pthread -ldl */
#include <Buzzer.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>

using namespace std;
using namespace CrossControl;

// 'Beep' implementation
void MyBeep(BUZZERHANDLE pBuzzer, unsigned short freq, int duration)
{
    CrossControl::eErr err;

    if(!pBuzzer)
        return;
    err = pBuzzer->setFrequency(freq);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function setFrequency: " << GetErrorStringA
            (err) << endl;
    }
    else
    {
        err = pBuzzer->buzze(duration, true);
        if(err != ERR_SUCCESS)
        {
            cout << "Error(" << err << ") in function buzze: " << GetErrorStringA(err
                ) << endl;
        }
    }
}

void play_beeps(BUZZERHANDLE pBuzzer)
{
    if(!pBuzzer)
```



```

        return;

    CrossControl::eErr err;
    unsigned short vol;

    err = pBuzzer->getVolume(&vol);
    if(err == ERR_SUCCESS)
    {
        cout << "Buzzer volume was: " << vol << endl;
    }
    else
    {
        cout << "Error(" << err << ") in function getVolume: " << GetErrorStringA(
            err) << endl;
        vol = 40;
    }

    err = pBuzzer->setVolume(20);
    if(err == ERR_SUCCESS)
    {
        cout << "Buzzer volume set to 20" << endl;
    }
    else
    {
        cout << "Error(" << err << ") in function setVolume: " << GetErrorStringA(
            err) << endl;
    }

    MyBeep(pBuzzer, 1000, 500);
    err = pBuzzer->setVolume(40);
    MyBeep(pBuzzer, 900, 500);
    err = pBuzzer->setVolume(51);
    MyBeep(pBuzzer, 700, 500);

    cout << "Restoring volume: " << vol << endl;
    err = pBuzzer->setVolume(vol);
}

int main(void)
{
    BUZZERHANDLE pBuzzer = ::GetBuzzer();
    assert(pBuzzer);

    play_beeps(pBuzzer);

    pBuzzer->Release();
}

```

5.5.2 Member Function Documentation

5.5.2.1 virtual eErr CrossControl::Buzzer::buzze (int *time*, bool *blocking*) [pure virtual]

Buzzes for a specified time.

Parameters

<i>time</i>	Time (ms) to buzz.
<i>blocking</i>	Blocking or non-blocking function.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pBuzzer->setFrequency(freq);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setFrequency: " << GetErrorStringA
        (err) << endl;
}
else
{
    err = pBuzzer->buzze(duration, true);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function buzze: " << GetErrorStringA(err
            ) << endl;
    }
}
```

5.5.2.2 virtual eErr CrossControl::Buzzer::getFrequency (unsigned short * *frequency*) [pure virtual]

Get buzzer frequency.

Parameters

<i>frequency</i>	Current frequency (700-10000 Hz).
------------------	-----------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.5.2.3 virtual eErr CrossControl::Buzzer::getTrigger (bool * *trigger*) [pure virtual]

Get buzzer trigger. The [Buzzer](#) is enabled when the trigger is enabled.

Parameters

<i>trigger</i>	Current trigger status.
----------------	-------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.5.2.4 virtual eErr CrossControl::Buzzer::getVolume (unsigned short * *volume*) [pure virtual]

Get buzzer volume.

Parameters

<i>volume</i>	Current volume (0-51).
---------------	------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pBuzzer->getVolume(&vol);
if(err == ERR_SUCCESS)
{
    cout << "Buzzer volume was: " << vol << endl;
}
else
{
    cout << "Error(" << err << ") in function getVolume: " << GetErrorStringA(
        err) << endl;
    vol = 40;
}
```

5.5.2.5 virtual void CrossControl::Buzzer::Release () [pure virtual]

Delete the [Buzzer](#) object.

Returns

-

Example Usage:

```
BUZZERHANDLE pBuzzer = ::GetBuzzer();
assert(pBuzzer);

play_beeps(pBuzzer);

pBuzzer->Release();
```

5.5.2.6 virtual eErr CrossControl::Buzzer::setFrequency (unsigned short *frequency*) [pure virtual]

Set buzzer frequency.

Parameters

<i>frequency</i>	Frequency to set (700-10000 Hz).
------------------	----------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pBuzzer->setFrequency(freq);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setFrequency: " << GetErrorStringA
        (err) << endl;
}
else
{
    err = pBuzzer->buzze(duration, true);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function buzze: " << GetErrorStringA(err
        ) << endl;
    }
}
```

5.5.2.7 virtual eErr CrossControl::Buzzer::setTrigger (bool *trigger*) [pure virtual]

Set buzzer trigger. The [Buzzer](#) is enabled when the trigger is enabled.

Parameters

<i>trigger</i>	Status to set.
----------------	----------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.5.2.8 virtual eErr CrossControl::Buzzer::setVolume (unsigned short *volume*) [pure virtual]

Set buzzer volume.

Parameters

<i>volume</i>	Volume to set (0-51).
---------------	-----------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pBuzzer->setVolume(20);
if(err == ERR_SUCCESS)
{
    cout << "Buzzer volume set to 20" << endl;
}
else
{
    cout << "Error(" << err << ") in function setVolume: " << GetErrorStringA(
        err) << endl;
}
```

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Buzzer.h](#)

5.6 CrossControl::BuzzerSetup Struct Reference

```
#include <CCAuxTypes.h>
```

Public Attributes

- unsigned short [frequency](#)
- unsigned short [volume](#)

5.6.1 Member Data Documentation

5.6.1.1 unsigned short CrossControl::BuzzerSetup::frequency

buzzer frequency

5.6.1.2 unsigned short CrossControl::BuzzerSetup::volume

buzzer volume

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/CCAuxTypes.h](#)

5.7 CrossControl::CanSetting Struct Reference

```
#include <CanSetting.h>
```

Public Member Functions

- virtual [eErr getBaudrate](#) (unsigned char net, unsigned short *baudrate)=0
- virtual [eErr getFrameType](#) (unsigned char net, [CanFrameType](#) *frameType)=0
- virtual [eErr setBaudrate](#) (unsigned char net, unsigned short baudrate)=0
- virtual [eErr setFrameType](#) (unsigned char net, [CanFrameType](#) frameType)=0
- virtual void [Release](#) ()=0

5.7.1 Detailed Description

Can settings

Use the globally defined function [GetCanSetting\(\)](#) to get a handle to the [CanSetting](#) struct. Use the method [CanSetting::Release\(\)](#) to return the handle.

Example Usage:

```

/* g++ -DLINUX examples/cansetting_example.cpp -lcc-aux -pthread -ldl */
#include <CanSetting.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>

using namespace std;
using namespace CrossControl;

void read_cansettings(CANSETTINGHANDLE pCanSetting)
{
    CrossControl::eErr err;
    unsigned short baudrates[4];
    CrossControl::CanFrameType frametypes[4];
    unsigned char net;

    if(!pCanSetting)
        return;

    for(net = 1; net <= 4; net++)
    {
        err = pCanSetting->getBaudrate(net, &baudrates[net-1]);
        if(err != ERR_SUCCESS)
        {
            cout << "Error(" << err << ") in function getBaudrate: " <<
                GetErrorStringA(err) << endl;
            break;
        }
    }

#ifdef LINUX

    err = pCanSetting->getFrameType(net, &frametypes[net-1]);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function getFrameType: " <<
            GetErrorStringA(err) << endl;
        break;
    }
#endif
}

```

```

    if(err == ERR_SUCCESS)
    {
        for(net = 1; net <= 4; net++)
        {
            cout << "Can" << (int)net << ": " << (int)baudrates[net-1] << "kbit/s";
#ifdef LINUX
            switch(frametypes[net-1])
            {
                case FrameStandard: cout << ", Standard" << endl; break;
                case FrameExtended: cout << ", Extended" << endl; break;
                case FrameStandardExtended: cout << ", Standard/Extended" << endl; break;
                default: cout << ", Undefined Frametype" << endl; break;
            }
#endif
        }
    }
}

int main(void)
{
    CANSETTINGHANDLE pCanSetting = ::GetCanSetting();
    assert(pCanSetting);

    read_cansettings(pCanSetting);

    pCanSetting->Release();
}

```

5.7.2 Member Function Documentation

5.7.2.1 virtual eErr CrossControl::CanSetting::getBaudrate (unsigned char *net*, unsigned short * *baudrate*) [pure virtual]

Get Baud rate

Parameters

<i>net</i>	CAN net (1-4) to get settings for.
<i>baudrate</i>	CAN baud rate (kbit/s).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pCanSetting->getBaudrate(net, &baudrates[net-1]);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getBaudrate: " <<
    GetErrorStringA(err) << endl;
    break;
}

```

5.7.2.2 virtual eErr CrossControl::CanSetting::getFrameType (unsigned char *net*, CanFrameType * *frameType*) [pure virtual]

Get frame type

Parameters

<i>net</i>	CAN net (1-4) to get settings for.
<i>frameType</i>	CAN frame type

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pCanSetting->getFrameType(net, &frametypes[net-1]);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getFrameType: " <<
    GetErrorStringA(err) << endl;
    break;
}
```

5.7.2.3 virtual void CrossControl::CanSetting::Release () [pure virtual]

Delete the [CanSetting](#) object.

Returns

-

Example Usage:

```
CANSETTINGHANDLE pCanSetting = ::GetCanSetting();
assert(pCanSetting);

read_cansettings(pCanSetting);

pCanSetting->Release();
```

5.7.2.4 virtual eErr CrossControl::CanSetting::setBaudrate (unsigned char *net*, unsigned short *baudrate*) [pure virtual]

Set Baud rate. The changes will take effect after a restart.

Parameters

<i>net</i>	CAN net (1-4).
<i>baudrate</i>	CAN baud rate (kbit/s). The driver will calculate the best supported baud rate if it does not support the given baud rate. The maximum baud rate is 1000 kbit/s.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.7.2.5 `virtual eErr CrossControl::CanSetting::setFrameType (unsigned char net, CanFrameType frameType)` [pure virtual]

Set frame type. The changes will take effect after a restart.

Parameters

<i>net</i>	CAN net (1-4).
<i>frameType</i>	CAN frameType

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/CanSetting.h](#)

5.8 CrossControl::Config Struct Reference

```
#include <Config.h>
```

Public Member Functions

- `virtual eErr getStartupTriggerConfig (TriggerConf *configuration)=0`
- `virtual eErr getShortButtonPressAction (PowerAction *action)=0`
- `virtual eErr getLongButtonPressAction (PowerAction *action)=0`
- `virtual eErr getOnOffSigAction (PowerAction *action)=0`
- `virtual eErr getFrontBtnTrigTime (unsigned short *triggertime)=0`
- `virtual eErr getExtOnOffSigTrigTime (unsigned long *triggertime)=0`
- `virtual eErr getSuspendMaxTime (unsigned short *maxTime)=0`
- `virtual eErr getCanStartupPowerConfig (CCStatus *status)=0`
- `virtual eErr getVideoStartupPowerConfig (unsigned char *config)=0`
- `virtual eErr getExtFanStartupPowerConfig (CCStatus *status)=0`
- `virtual eErr getStartupVoltageConfig (double *voltage)=0`
- `virtual eErr getHeatingTempLimit (signed short *temperature)=0`
- `virtual eErr getPowerOnStartup (CCStatus *status)=0`
- `virtual eErr setStartupTriggerConfig (TriggerConf conf)=0`
- `virtual eErr setShortButtonPressAction (PowerAction action)=0`
- `virtual eErr setLongButtonPressAction (PowerAction action)=0`
- `virtual eErr setOnOffSigAction (PowerAction action)=0`

- virtual [eErr setFrontBtnTrigTime](#) (unsigned short triggertime)=0
- virtual [eErr setExtOnOffSigTrigTime](#) (unsigned long triggertime)=0
- virtual [eErr setSuspendMaxTime](#) (unsigned short maxTime)=0
- virtual [eErr setCanStartupPowerConfig](#) (CCStatus status)=0
- virtual [eErr setVideoStartupPowerConfig](#) (unsigned char config)=0
- virtual [eErr setExtFanStartupPowerConfig](#) (CCStatus status)=0
- virtual [eErr setStartupVoltageConfig](#) (double voltage)=0
- virtual [eErr setHeatingTempLimit](#) (signed short temperature)=0
- virtual [eErr setPowerOnStartup](#) (CCStatus status)=0
- virtual void [Release](#) ()=0

5.8.1 Detailed Description

Video channel 4 config

Configuration of various settings

Use the globally defined function [GetConfig\(\)](#) to get a handle to the [Config](#) struct. Use the method [Config::Release\(\)](#) to return the handle.

Example Usage:

```

/* g++ -DLINUX examples/config_example.cpp -lcc-aux -pthread -ldl */
#include <Config.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>

using namespace std;
using namespace CrossControl;

void conf_example(CONFIGHANDLE pConfig)
{
    CrossControl::eErr err;
    CrossControl::TriggerConf trig;

    if(!pConfig)
        return;

    err = pConfig->getStartupTriggerConfig(&trig);
    if(err == ERR_SUCCESS)
    {
        cout << "Start-up trigger is set to: ";
        switch(trig)
        {
            case Front_Button_Enabled: cout << "Front button only" << endl; break;
            case OnOff_Signal_Enabled: cout << "On/Off signal only" << endl; break;
            case Both_Button_And_Signal_Enabled: cout << "Front button or On/off signal"
                << endl; break;
            default: cout << "Error - Undefined StartupTrigger" << endl; break;
        }
    }
    else
    {
        cout << "Error(" << err << ") in function getStartupTriggerConfig: " <<
            GetErrorStringA(err) << endl;
    }
}

```

```

    }

    //
    // Set the action to suspend mode when the front panel button is pressed
    // (short time).
    //
    err = pConfig->setShortButtonPressAction(ActionSuspend);
    if(err == ERR_SUCCESS)
    {
        cout << "ShortButtonPressAction set to Suspend!" << endl;
    }
    else
    {
        cout << "Error(" << err << ") in function setShortButtonPressAction: " <<
            GetErrorStringA(err) << endl;
    }
}

int main(void)
{
    CONFIGHANDLE pConfig = ::GetConfig();
    assert(pConfig);

    conf_example(pConfig);

    pConfig->Release();
}

```

5.8.2 Member Function Documentation

5.8.2.1 virtual eErr CrossControl::Config::getCanStartupPowerConfig (CCStatus * *status*) [pure virtual]

Get Can power at startup configuration. The status of Can power at startup and at resume from suspended mode. At resume from suspend, this setting overrides the setting of the setCanPowerStatus function.

Parameters

<i>status</i>	Enabled/Disabled
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.2 virtual eErr CrossControl::Config::getExtFanStartupPowerConfig (CCStatus * *status*) [pure virtual]

Get External fan power at startup configuration. The status at startup and at resume from suspended mode. At resume from suspend, this setting overrides the setting of the setExtFanPowerStatus function.

Parameters

<i>status</i>	Enabled/Disabled
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.3 virtual eErr CrossControl::Config::getExtOnOffSigTrigTime (unsigned long * *triggertime*) [pure virtual]

Get external on/off signal trigger time.

Parameters

<i>triggertime</i>	Time in seconds that the external signal has to be low for the unit to enter suspend mode or shut down (trigger an action). This time can be set from one second up to several years, if needed.
--------------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.4 virtual eErr CrossControl::Config::getFrontBtnTrigTime (unsigned short * *triggertime*) [pure virtual]

Get front button trigger time for long press.

Parameters

<i>triggertime</i>	Time in milliseconds that the button has to be pressed for the press to count as a long button press. A button press twice this time will generate a hard shut down. If this time is set under 4000ms, the hard shut down minimum time of 8s is used instead.
--------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.5 virtual eErr CrossControl::Config::getHeatingTempLimit (signed short * *temperature*) [pure virtual]

Get the current limit for heating. When temperature is below this limit, the system is internally heated until the temperature rises above the limit. The default and minimum value is -25 degrees Celsius. The maximum value is +5 degrees Celsius.

Parameters

<i>temperature</i>	The current heating limit, in degrees Celsius (-25 to +5)
--------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.6 virtual eErr CrossControl::Config::getLongButtonPressAction (PowerAction * *action*) [pure virtual]

Get long button press action. Gets the configured action for a long button press: No-Action, ActionSuspend or ActionShutDown. A long button press is determined by the FrontBtnTrigTime.

Parameters

<i>action</i>	The configured action.
---------------	------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.7 virtual eErr CrossControl::Config::getOnOffSigAction (PowerAction * *action*) [pure virtual]

Get On/Off signal action. Gets the configured action for an On/Off signal event: No-Action, ActionSuspend or ActionShutDown. An On/Off signal event is determined by the ExtOnOffSigTrigTime.

Parameters

<i>action</i>	The configured action.
---------------	------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.8 virtual eErr CrossControl::Config::getPowerOnStartup (CCStatus * *status*) [pure virtual]

Get power on start-up behavior. If enabled, the unit always starts when power is turned on, disregarding the setting for StartupTriggerConfig at that time. The StartupTriggerConfig still applies if the unit is shut down or suspended, without removing the power supply.

Parameters

<i>status</i>	Enabled/Disabled
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.9 virtual eErr CrossControl::Config::getShortButtonPressAction (PowerAction * *action*) [pure virtual]

Get short button press action. Gets the configured action for a short button press: No-Action, ActionSuspend or ActionShutDown.

Parameters

<i>action</i>	The configured action.
---------------	------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.10 virtual eErr CrossControl::Config::getStartupTriggerConfig (TriggerConf * *configuration*) [pure virtual]

Get Start-up trigger configuration. Is the front button and/or the external on/off signal enabled as triggers for startup and wake up from suspended mode?

Parameters

<i>configuration</i>	One of: Front_Button_Enabled, OnOff_Signal_Enabled or Both_Button_And_Signal_Enabled.
----------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pConfig->getStartupTriggerConfig(&trig);
if(err == ERR_SUCCESS)
{
    cout << "Start-up trigger is set to: ";
    switch(trig)
    {
        case Front_Button_Enabled: cout << "Front button only" << endl; break;
        case OnOff_Signal_Enabled: cout << "On/Off signal only" << endl; break;
        case Both_Button_And_Signal_Enabled: cout << "Front button or On/off signal"
            << endl; break;
        default: cout << "Error - Undefined StartupTrigger" << endl; break;
    }
}
```

```

    }
}
else
{
    cout << "Error(" << err << ") in function getStartupTriggerConfig: " <<
        GetErrorStringA(err) << endl;
}

```

5.8.2.11 virtual eErr CrossControl::Config::getStartupVoltageConfig (double * *voltage*) [pure virtual]

Get the voltage threshold required for startup. The external voltage must be stable above this value for the unit to start up. The default and minimum value is 9V. It could be set to a higher value for a 24V system.

Parameters

<i>voltage</i>	The current voltage setting. (9V .. 28V)
----------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.12 virtual eErr CrossControl::Config::getSuspendMaxTime (unsigned short * *maxTime*) [pure virtual]

Get suspend mode maximum time.

Parameters

<i>maxTime</i>	Maximum suspend time in minutes. After this time in suspended mode, the unit will shut down to save power. A value of 0 means that the automatic shut down function is not used.
----------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.13 virtual eErr CrossControl::Config::getVideoStartupPowerConfig (unsigned char * *config*) [pure virtual]

Get [Video](#) power at startup configuration. The status of [Video](#) power at startup and at resume from suspended mode. At resume from suspend, this setting overrides the setting of the setVideoPowerStatus function.

Parameters

<i>config</i>	Bitwise representation of the four video channels. See the VideoXConf defines. if the bit is 1, the power is enabled, else disabled.
---------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.14 virtual void CrossControl::Config::Release() [pure virtual]

Delete the [Config](#) object.

Returns

-

Example Usage:

```
CONFIGHANDLE pConfig = ::GetConfig();
assert(pConfig);

conf_example(pConfig);

pConfig->Release();
```

5.8.2.15 virtual eErr CrossControl::Config::setCanStartupPowerConfig (CCStatus *status*) [pure virtual]

Set Can power at startup configuration. The status of Can power at startup and at resume from suspended mode. At resume from suspend, this setting overrides the setting of the setCanPowerStatus function.

Parameters

<i>status</i>	Enabled/Disabled
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.16 virtual eErr CrossControl::Config::setExtFanStartupPowerConfig (CCStatus *status*) [pure virtual]

Set External fan power at startup configuration. The status at startup and at resume from suspended mode. At resume from suspend, this setting overrides the setting of the setExtFanPowerStatus function.

Parameters

<i>status</i>	Enabled/Disabled
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.17 virtual eErr CrossControl::Config::setExtOnOffSigTrigTime (unsigned long *triggertime*) [pure virtual]

Set external on/off signal trigger time.

Parameters

<i>triggertime</i>	Time in seconds that the external signal has to be low for the unit to enter suspend mode or shut down (trigger an action). This time can be set from one second up to several years, if needed.
--------------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.18 virtual eErr CrossControl::Config::setFrontBtnTrigTime (unsigned short *triggertime*) [pure virtual]

Set front button trigger time for long press.

Parameters

<i>triggertime</i>	Time in milliseconds that the button has to be pressed for the press to count as a long button press. A button press twice this time will generate a hard shut down. If this time is set under 4000ms, the hard shut down minimum time of 8s is used instead.
--------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.19 virtual eErr CrossControl::Config::setHeatingTempLimit (signed short *temperature*) [pure virtual]

Set the current limit for heating. When temperature is below this limit, the system is internally heated until the temperature rises above the limit. The default and minimum value is -25 degrees Celsius. The maximum value is +5 degrees Celsius.

Parameters

<i>temperature</i>	The heating limit, in degrees Celsius (-25 to +5)
--------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.20 virtual eErr CrossControl::Config::setLongButtonPressAction (PowerAction *action*) [pure virtual]

Set long button press action. Sets the configured action for a long button press: No-Action, ActionSuspend or ActionShutDown. A long button press is determined by the FrontBtnTrigTime.

Parameters

<i>action</i>	The action to set.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.21 virtual eErr CrossControl::Config::setOnOffSigAction (PowerAction *action*) [pure virtual]

Set On/Off signal action. Sets the configured action for an On/Off signal event: No-Action, ActionSuspend or ActionShutDown. An On/Off signal event is determined by the ExtOnOffSigTrigTime.

Parameters

<i>action</i>	The action to set.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.22 virtual eErr CrossControl::Config::setPowerOnStartup (CCStatus *status*) [pure virtual]

Set power on start-up behavior. If enabled, the unit always starts when power is turned on, disregarding the setting for StartupTriggerConfig at that time. The StartupTriggerConfig still applies if the unit is shut down or suspended, without removing the power supply.

Parameters

<i>status</i>	Enabled/Disabled
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.23 virtual eErr CrossControl::Config::setShortButtonPressAction (PowerAction *action*) [pure virtual]

Set short button press action. Sets the configured action for a short button press: No-Action, ActionSuspend or ActionShutDown.

Parameters

<i>action</i>	The action to set.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pConfig->setShortButtonPressAction(ActionSuspend);
if(err == ERR_SUCCESS)
{
    cout << "ShortButtonPressAction set to Suspend!" << endl;
}
else
{
    cout << "Error(" << err << ") in function setShortButtonPressAction: " <<
        GetErrorStringA(err) << endl;
}
```

5.8.2.24 virtual eErr CrossControl::Config::setStartupTriggerConfig (TriggerConf *conf*) [pure virtual]

Set Start-up trigger configuration. Should the front button and/or the external on/off signal be enabled as triggers for startup and wake up from suspended mode?

Parameters

<i>conf</i>	Must be one of: Front_Button_Enabled, OnOff_Signal_Enabled or - Both_Button_And_Signal_Enabled.
-------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.25 `virtual eErr CrossControl::Config::setStartupVoltageConfig (double voltage)` [pure virtual]

Set the voltage threshold required for startup. The external voltage must be stable above this value for the unit to start up. The default and minimum value is 9V. It could be set to a higher value for a 24V system.

Parameters

<i>voltage</i>	The voltage to set (9V .. 28V).
----------------	---------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.26 `virtual eErr CrossControl::Config::setSuspendMaxTime (unsigned short maxTime)` [pure virtual]

Set suspend mode maximum time.

Parameters

<i>maxTime</i>	Maximum suspend time in minutes. After this time in suspended mode, the unit will shut down to save power. A value of 0 means that this function is not used.
----------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.8.2.27 `virtual eErr CrossControl::Config::setVideoStartupPowerConfig (unsigned char config)` [pure virtual]

Set [Video](#) power at startup configuration. The status of [Video](#) power at startup and at resume from suspended mode. At resume from suspend, this setting overrides the setting of the `setVideoPowerStatus` function.

Parameters

<i>config</i>	Bitwise representation of the four video channels. See the <code>VideoXConf</code> defines. if the bit is 1, the power is enabled, else disabled.
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Config.h](#)

5.9 CrossControl::Diagnostic Struct Reference

```
#include <Diagnostic.h>
```

Public Member Functions

- virtual [eErr getSSTemp](#) (signed short *temperature)=0
- virtual [eErr getPCBTemp](#) (signed short *temperature)=0
- virtual [eErr getPMTemp](#) (unsigned char index, signed short *temperature, [Jida-SensorType](#) *jst)=0
- virtual [eErr getStartupReason](#) (unsigned short *reason)=0
- virtual [eErr getShutdownReason](#) (unsigned short *reason)=0
- virtual [eErr getHwErrorStatus](#) (unsigned short *errorCode)=0
- virtual [eErr getTimer](#) ([TimerType](#) *times)=0
- virtual [eErr getMinMaxTemp](#) (signed short *minTemp, signed short *maxTemp)=0
- virtual [eErr getPowerCycles](#) (unsigned short *powerCycles)=0
- virtual [eErr clearHwErrorStatus](#) (void)=0
- virtual void [Release](#) ()=0

5.9.1 Detailed Description

Access to unit diagnostic data

Use the globally defined function [GetDiagnostic\(\)](#) to get a handle to the [Diagnostic](#) struct. Use the method [Diagnostic::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/diagnostic_example.cpp -lcc-aux -pthread -ldl */
#include <Diagnostic.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace CrossControl;

void printStringTime(eErr err, char* text, unsigned long value)
{
    unsigned long rest;
    unsigned day, hour;

    day = value / 1440;
    rest = value % 1440;
    hour = rest / 60;
    rest = value % 60;
```

```
    if(err == ERR_SUCCESS)
    {
        printf("%s: %d min (%d days, %dh, %dmin)\n", text, value, day, hour, rest);
    }
    else
    {
        printf("%s: Error(%d) %s\n", text, err, CrossControl::GetErrorStringA(err))
            ;
    }
}

void printString(eErr err, char* text, signed short value, char* text2)
{
    if(err == ERR_SUCCESS)
    {
        printf("%s: %d %s\n", text, value, text2);
    }
    else
    {
        printf("%s: Error(%d) %s\n", text, err, CrossControl::GetErrorStringA(err))
            ;
    }
}

void diagnostic_example(DIAGNOSTICHANDLE pDiagnostic)
{
    CrossControl::eErr err;
    TimerType tt;
    signed short sValue, sValue2;

    if(!pDiagnostic)
        return;

    err = pDiagnostic->getSSTemp(&sValue);
    printString(err, "Main board (SS) temp", sValue, "deg C");

    err = pDiagnostic->getTimer(&tt);
    printStringTime(err, "Total run time", tt.TotRunTime);
    printStringTime(err, "Total suspend time", tt.TotSuspTime);
    printStringTime(err, "Total heat time", tt.TotHeatTime);
    printStringTime(err, "Total run time 40-60 deg C", tt.RunTime40_60);
    printStringTime(err, "Total run time 60-70 deg C", tt.RunTime60_70);
    printStringTime(err, "Total run time 70-80 deg C", tt.RunTime70_80);
    printStringTime(err, "Total run time above 80 deg C", tt.Above80RunTime);

    err = pDiagnostic->getMinMaxTemp(&sValue, &sValue2);
    printString(err, "Minimum temp", sValue, "deg C");
    printString(err, "Maximum temp", sValue2, "deg C");
}

int main(void)
{
    DIAGNOSTICHANDLE pDiagnostic = ::GetDiagnostic();
    assert(pDiagnostic);

    diagnostic_example(pDiagnostic);

    pDiagnostic->Release();
}
```

5.9.2 Member Function Documentation

5.9.2.1 virtual eErr CrossControl::Diagnostic::clearHwErrorStatus (void) [pure virtual]

Clear the HW error status (this function is used by the [CrossControl](#) service/daemon to log any hardware errors)

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.9.2.2 virtual eErr CrossControl::Diagnostic::getHwErrorStatus (unsigned short * *errorCode*) [pure virtual]

Get hardware error code. If hardware errors are found or other problems are discovered by the SS, they are reported here. See [DiagnosticCodes.h](#) for error codes.

Parameters

<i>errorCode</i>	Error code. Zero means no error.
------------------	----------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.9.2.3 virtual eErr CrossControl::Diagnostic::getMinMaxTemp (signed short * *minTemp*, signed short * *maxTemp*) [pure virtual]

Get diagnostic temperature interval of the unit.

Parameters

<i>minTemp</i>	Minimum measured PCB temperature.
<i>maxTemp</i>	Maximum measured PCB temperature.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pDiagnostic->getMinMaxTemp(&sValue, &sValue2);
printString(err, "Minimum temp", sValue, "deg C");
printString(err, "Maximum temp", sValue2, "deg C");
```

5.9.2.4 virtual eErr CrossControl::Diagnostic::getPCBTemp (signed short * *temperature*) [pure virtual]

Get PCB temperature.

Parameters

<i>temperature</i>	PCB Temperature in degrees Celsius.
--------------------	-------------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.9.2.5 virtual eErr CrossControl::Diagnostic::getPMTemp (unsigned char *index*, signed short * *temperature*, JidaSensorType * *jst*) [pure virtual]

Get Processor Module temperature. This temperature is read from the Kontron JIDA API. This API also has a number of other functions, please see the JIDA documentation for how to use them separately.

Parameters

<i>index</i>	Zero-based index of the temperature sensor. Different boards may have different number of sensors. The CCpilot XM currently has 2 sensors, board and cpu. An error is returned if the index is not supported.
<i>temperature</i>	Temperature in degrees Celsius.
<i>jst</i>	The type of sensor that is being read.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.9.2.6 virtual eErr CrossControl::Diagnostic::getPowerCycles (unsigned short * *powerCycles*) [pure virtual]

Get number of power cycles.

Parameters

<i>powerCycles</i>	Total number of power cycles.
--------------------	-------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.9.2.7 virtual eErr CrossControl::Diagnostic::getShutdownReason (unsigned short * *reason*) [pure virtual]

Get shutdown reason.

Parameters

<i>reason</i>	See DiagnosticCodes.h for shutdown codes.
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.9.2.8 virtual eErr CrossControl::Diagnostic::getSSTemp (signed short * *temperature*) [pure virtual]

Get System Supervisor temperature.

Parameters

<i>temperature</i>	System Supervisor temperature in degrees Celsius.
--------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pDiagnostic->getSSTemp(&sValue);
printString(err, "Main board (SS) temp", sValue, "deg C");
```

5.9.2.9 virtual eErr CrossControl::Diagnostic::getStartupReason (unsigned short * *reason*) [pure virtual]

Get startup reason.

Parameters

<i>reason</i>	See DiagnosticCodes.h for startup codes.
---------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.9.2.10 virtual eErr CrossControl::Diagnostic::getTimer (TimerType * times) [pure virtual]

Get diagnostic timer.

Parameters

<i>times</i>	Get a struct with the current diagnostic times.
--------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pDiagnostic->getTimer(&tt);
printStringTime(err, "Total run time", tt.TotRunTime);
printStringTime(err, "Total suspend time", tt.TotSuspTime);
printStringTime(err, "Total heat time", tt.TotHeatTime);
printStringTime(err, "Total run time 40-60 deg C", tt.RunTime40_60);
printStringTime(err, "Total run time 60-70 deg C", tt.RunTime60_70);
printStringTime(err, "Total run time 70-80 deg C", tt.RunTime70_80);
printStringTime(err, "Total run time above 80 deg C", tt.Above80RunTime);
```

5.9.2.11 virtual void CrossControl::Diagnostic::Release () [pure virtual]

Delete the [Diagnostic](#) object.

Returns

-

Example Usage:

```
DIAGNOSTICHANDLE pDiagnostic = ::GetDiagnostic();
assert(pDiagnostic);

diagnostic_example(pDiagnostic);

pDiagnostic->Release();
```

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Diagnostic.h](#)

5.10 CrossControl::DigIO Struct Reference

```
#include <DigIO.h>
```

Public Member Functions

- virtual [eErr getDigIO](#) (unsigned char *status)=0
- virtual [eErr setDigIO](#) (unsigned char state)=0
- virtual [eErr setDigIODir](#) (unsigned char dir)=0
- virtual [eErr getDigIODir](#) (unsigned char *dir)=0
- virtual void [Release](#) ()=0

5.10.1 Detailed Description

Read digital inputs

Use the globally defined function [GetDigIO\(\)](#) to get a handle to the [DigIO](#) struct. Use the method [DigIO::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/digio_example.cpp -lcc-aux -pthread -ldl */
#include <assert.h>
#include <DigIO.h>
#include <iostream>

using namespace std;

void list_digital_inputs(DIGIOHANDLE pDigIO)
{
    if(!pDigIO)
        return;

    CrossControl::eErr err;
    unsigned char inputs;

    err = pDigIO->getDigIO (&inputs);
    if (CrossControl::ERR_SUCCESS == err)
    {
        cout << "Digital In 1: " <<
            ((inputs & CrossControl::DigitalIn_1) ? "High" : "Low") << endl;
        cout << "Digital In 2: " <<
            ((inputs & CrossControl::DigitalIn_2) ? "High" : "Low") << endl;
        cout << "Digital In 3: " <<
            ((inputs & CrossControl::DigitalIn_3) ? "High" : "Low") << endl;
        cout << "Digital In 4: " <<
            ((inputs & CrossControl::DigitalIn_4) ? "High" : "Low") << endl;
    }
    else
    {
        cout << "Unable to read digital input status." << endl;
    }
}

int main(void)
{
    DIGIOHANDLE pDigIO = ::GetDigIO();
    assert(pDigIO);

    list_digital_inputs(pDigIO);

    pDigIO->Release();
}
```

```
}

```

5.10.2 Member Function Documentation

5.10.2.1 virtual eErr CrossControl::DigIO::getDigIO (unsigned char * *status*) [pure virtual]

Get Digital inputs.

Parameters

<i>status</i>	Status of the four digital input pins. Bit0: Digital input 1. Bit1: Digital input 2. Bit2: Digital input 3. Bit3: Digital input 4. Bit 4..7 are always zero.
---------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pDigIO->getDigIO (&inputs);
if (CrossControl::ERR_SUCCESS == err)
{
    cout << "Digital In 1: " <<
        ((inputs & CrossControl::DigitalIn_1) ? "High" : "Low") << endl;
    cout << "Digital In 2: " <<
        ((inputs & CrossControl::DigitalIn_2) ? "High" : "Low") << endl;
    cout << "Digital In 3: " <<
        ((inputs & CrossControl::DigitalIn_3) ? "High" : "Low") << endl;
    cout << "Digital In 4: " <<
        ((inputs & CrossControl::DigitalIn_4) ? "High" : "Low") << endl;
}
else
{
    cout << "Unable to read digital input status." << endl;
}

```

5.10.2.2 virtual eErr CrossControl::DigIO::getDigIODir (unsigned char * *dir*) [pure virtual]

Get Digital io direction.

Parameters

<i>dir</i>	Direction of the four digital io pins. Bit0: Digital io 1 dir (0 = input, 1 = output). Bit1: Digital io 2 dir (0 = input, 1 = output). Bit2: Digital io 3 dir (0 = input, 1 = output). Bit3: Digital io 4 dir (0 = input, 1 = output). Bit 4..7 are always zero.
------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

5.10.2.3 virtual void CrossControl::DigIO::Release () [pure virtual]

Delete the [DigIO](#) object.

Returns

-

Example Usage:

```
DIGIOHANDLE pDigIO = ::GetDigIO();
assert(pDigIO);

list_digital_inputs(pDigIO);

pDigIO->Release();
```

5.10.2.4 virtual eErr CrossControl::DigIO::setDigIO (unsigned char *state*) [pure virtual]

Set Digital outputs.

Parameters

<i>state</i>	State of the four digital output pins. Bit0: Digital output 1. Bit1: - Digital output 2. Bit2: Digital output 3. Bit3: Digital output 4. Bit 4..7 are always zero.
--------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

5.10.2.5 virtual eErr CrossControl::DigIO::setDigIODir (unsigned char *dir*) [pure virtual]

Set Digital io direction.

Parameters

<i>dir</i>	Direction of the four digital io pins. Bit0: Digital io 1 dir (0 = input, 1 = output). Bit1: Digital io 2 dir (0 = input, 1 = output). Bit2: Digital io 3 dir (0 = input, 1 = output). Bit3: Digital io 4 dir (0 = input, 1 = output). Bit 4..7 are always zero.
------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/DigIO.h](#)

5.11 CrossControl::FirmwareUpgrade Struct Reference

```
#include <FirmwareUpgrade.h>
```

Public Member Functions

- virtual [eErr startFpgaUpgrade](#) (const char *filename, bool blocking)=0
- virtual [eErr startFpgaVerification](#) (const char *filename, bool blocking)=0
- virtual [eErr startSSUpgrade](#) (const char *filename, bool blocking)=0
- virtual [eErr startSSVerification](#) (const char *filename, bool blocking)=0
- virtual [eErr startFrontUpgrade](#) (const char *filename, bool blocking)=0
- virtual [eErr startFrontVerification](#) (const char *filename, bool blocking)=0
- virtual [eErr getUpgradeStatus](#) ([UpgradeStatus](#) *status, bool blocking)=0
- virtual [eErr shutDown](#) ()=0
- virtual void [Release](#) ()=0

5.11.1 Detailed Description

Firmware upgrade of the system's microprocessors and FPGA.

Use the globally defined function [GetFirmwareUpgrade\(\)](#) to get a handle to the [FirmwareUpgrade](#) struct. Use the method [FirmwareUpgrade::Release\(\)](#) to return the handle.

Example Usage:

```
#include <assert.h>
#include <FirmwareUpgrade.h>
#include <iostream>
#ifdef LINUX
#include <cstring>
```

```
#else
#include <string>
#endif

using namespace std;

void upgrade_ss(string path)
{
    const int max_retries = 3;
    CrossControl::eErrr err;

    FIRMWAREUPGHANDLE upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    cout << "Upgrading SS" << endl;

    for(int i=0;i<max_retries;i++)
    {
        // Reinitialize upgrade handle
        upgrade->Release();
        upgrade=GetFirmwareUpgrade();
        assert(upgrade != NULL);

        err = upgrade->startSSUpgrade(path.c_str(), true);
        if (CrossControl::ERR_SUCCESS == err) {
            cout << "Upgrade Ok" << endl;
            break;
        }
        else if(CrossControl::ERR_VERIFY_FAILED == err) {
            // Reinitialize upgrade handle
            upgrade->Release();
            upgrade=GetFirmwareUpgrade();
            assert(upgrade != NULL);

            err = upgrade->startSSVerification(path.c_str(), true);

            if (CrossControl::ERR_SUCCESS == err) {
                cout << "Upgrade Ok" << endl;
                break;
            }
        }
    }

    upgrade->Release();
}

void upgrade_front(string path)
{
    const int max_retries = 3;
    CrossControl::eErrr err;

    FIRMWAREUPGHANDLE upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    cout << "Upgrading front" << endl;

    for(int i=0;i<max_retries;i++)
    {
        // Reinitialize upgrade handle
        upgrade->Release();
        upgrade=GetFirmwareUpgrade();
        assert(upgrade != NULL);
```

```
err = upgrade->startFrontUpgrade(path.c_str(), true);
if (CrossControl::ERR_SUCCESS == err) {
    cout << "Upgrade Ok" << endl;
    break;
}
else if(CrossControl::ERR_VERIFY_FAILED == err) {
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    err = upgrade->startFrontVerification(path.c_str(), true);

    if (CrossControl::ERR_SUCCESS == err) {
        cout << "Upgrade Ok" << endl;
        break;
    }
}
}

upgrade->Release();
}

void upgrade_fpga(string path)
{
    const int max_retries = 3;
    CrossControl::eErr err;

    FIRMWAREUPGHANDLE upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    cout << "Upgrading FPGA" << endl;

    for(int i=0;i<max_retries;i++)
    {
        // Reinitialize upgrade handle
        upgrade->Release();
        upgrade=GetFirmwareUpgrade();
        assert(upgrade != NULL);

        err = upgrade->startFpgaUpgrade(path.c_str(), true);
        if (CrossControl::ERR_SUCCESS == err) {
            cout << "Upgrade Ok" << endl;
            break;
        }
        else if(CrossControl::ERR_VERIFY_FAILED == err) {
            // Reinitialize upgrade handle
            upgrade->Release();
            upgrade=GetFirmwareUpgrade();
            assert(upgrade != NULL);

            err = upgrade->startFpgaVerification(path.c_str(), true);

            if (CrossControl::ERR_SUCCESS == err) {
                cout << "Upgrade Ok" << endl;
                break;
            }
        }
    }

    upgrade->Release();
}
```



```

}

void usage(string progname)
{
    cout << "Usage: " << progname <<
        " <fpga|ss|front> <path>" << endl;
}

int main(int argc, char *argv[])
{
    if(argc != 3) {
        usage(argv[0]);
        return -1;
    }

    if(!strcmp(argv[1], "fpga"))
        upgrade_fpga(argv[2]);
    else if(!strcmp(argv[1], "ss"))
        upgrade_ss(argv[2]);
    else if(!strcmp(argv[1], "front"))
        upgrade_front(argv[2]);
    else
        usage(argv[0]);

    return 0;
}

```

5.11.2 Member Function Documentation

5.11.2.1 virtual eErr CrossControl::FirmwareUpgrade::getUpgradeStatus (UpgradeStatus * status, bool blocking) [pure virtual]

Gets the status of an upgrade operation. The upgrade status is common for all upgrade and verification methods.

Parameters

<i>status</i>	The current status of the upgrade operation.
<i>blocking</i>	Whether or not the function should wait until a new status event has been reported. If blocking is set to false, the function will return immediately with the current status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.11.2.2 virtual void CrossControl::FirmwareUpgrade::Release () [pure virtual]

Delete the [FirmwareUpgrade](#) object.

Returns

-

Example Usage:

```
upgrade->Release();
```

5.11.2.3 virtual eErr CrossControl::FirmwareUpgrade::shutDown () [pure virtual]

Shut down the operating system.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.11.2.4 virtual eErr CrossControl::FirmwareUpgrade::startFpgaUpgrade (const char * filename, bool blocking) [pure virtual]

Start an upgrade of the FPGA. After a FPGA upgrade, the system should be shut down. Full functionality of the system cannot be guaranteed until a fresh startup has been performed.

Note that if you intend to do several upgrades/verifications in a row, the [FirmwareUpgrade](#) object should be released and reinitialised between each operation: `pFirmwareUpgrade->Release(); pFirmwareUpgrade = GetFirmwareUpgrade();`

Parameters

<i>filename</i>	Path and filename to the .mcs file to program.
<i>blocking</i>	Whether or not the function should wait until completion. If blocking is set to false, the function will return immediately. One must then call <code>getUpgradeStatus</code> to get the status of the upgrade operation. If blocking is set to true, the function will return when the operation is complete. This might take a few minutes.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
cout << "Upgrading FPGA" << endl;

for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
}
```

```

assert(upgrade != NULL);

err = upgrade->startFpgaUpgrade(path.c_str(), true);
if (CrossControl::ERR_SUCCESS == err) {
    cout << "Upgrade Ok" << endl;
    break;
}
else if(CrossControl::ERR_VERIFY_FAILED == err) {
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    err = upgrade->startFpgaVerification(path.c_str(), true);

    if (CrossControl::ERR_SUCCESS == err) {
        cout << "Upgrade Ok" << endl;
        break;
    }
}
}
}

```

5.11.2.5 virtual eErr CrossControl::FirmwareUpgrade::startFpgaVerification (const char * filename, bool blocking) [pure virtual]

Start a verification of the FPGA. Verifies the FPGA against the file to program. This could be useful if verification during programming fails.

Note that if you intend to do several upgrades/verifications in a row, the [FirmwareUpgrade](#) object should be released and reinitialised between each operation: pFirmwareUpgrade->[Release\(\)](#); pFirmwareUpgrade = [GetFirmwareUpgrade\(\)](#);

Parameters

<i>filename</i>	Path and filename to the .mcs file to verify against.
<i>blocking</i>	Whether or not the function should wait until completion. If blocking is set to false, the function will return immediately. One must then call getUpgradeStatus to get the status of the operation. If blocking is set to true, the function will return when the operation is complete. This might take a few minutes.

Returns

error status. 0 = [ERR_SUCCESS](#), otherwise error code.

Example Usage:

```

cout << "Upgrading FPGA" << endl;

for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);
}

```

```

err = upgrade->startFpgaUpgrade(path.c_str(), true);
if (CrossControl::ERR_SUCCESS == err) {
    cout << "Upgrade Ok" << endl;
    break;
}
else if(CrossControl::ERR_VERIFY_FAILED == err) {
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    err = upgrade->startFpgaVerification(path.c_str(), true);

    if (CrossControl::ERR_SUCCESS == err) {
        cout << "Upgrade Ok" << endl;
        break;
    }
}
}
}

```

5.11.2.6 virtual eErr CrossControl::FirmwareUpgrade::startFrontUpgrade (const char * filename, bool blocking) [pure virtual]

Start an upgrade of the front microprocessor. After a front upgrade, the system should be shut down. The front will not work until a fresh startup has been performed.

Note that if you intend to do several upgrades/verifications in a row, the [Firmware-Upgrade](#) object should be released and reinitialised between each operation: pFirmwareUpgrade->[Release\(\)](#); pFirmwareUpgrade = [GetFirmwareUpgrade\(\)](#);

Parameters

<i>filename</i>	Path and filename to the .hex file to program.
<i>blocking</i>	Whether or not the function should wait until completion. If blocking is set to false, the function will return immediately. One must then call <code>fpgaUpgradeStatus</code> to get the status of the upgrade operation. If blocking is set to true, the function will return when the operation is complete. This might take a few minutes.

Returns

error status. 0 = `ERR_SUCCESS`, otherwise error code.

Example Usage:

```

cout << "Upgrading front" << endl;

for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);
}

```

```

err = upgrade->startFrontUpgrade(path.c_str(), true);
if (CrossControl::ERR_SUCCESS == err) {
    cout << "Upgrade Ok" << endl;
    break;
}
else if(CrossControl::ERR_VERIFY_FAILED == err) {
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    err = upgrade->startFrontVerification(path.c_str(), true);

    if (CrossControl::ERR_SUCCESS == err) {
        cout << "Upgrade Ok" << endl;
        break;
    }
}
}
}

```

5.11.2.7 virtual eErr CrossControl::FirmwareUpgrade::startFrontVerification (const char * filename, bool blocking) [pure virtual]

Start a verification of the front microprocessor. Verifies the front microprocessor against the file to program. This could be useful if verification during programming fails.

Note that if you intend to do several upgrades/verifications in a row, the [Firmware-Upgrade](#) object should be released and reinitialised between each operation: pFirmwareUpgrade->[Release\(\)](#); pFirmwareUpgrade = [GetFirmwareUpgrade\(\)](#);

Parameters

<i>filename</i>	Path and filename to the .hex file to verify against.
<i>blocking</i>	Whether or not the function should wait until completion. If blocking is set to false, the function will return immediately. One must then call getUpgradeStatus to get the status of the operation. If blocking is set to true, the function will return when the operation is complete. This might take a few minutes.

Returns

error status. 0 = [ERR_SUCCESS](#), otherwise error code.

Example Usage:

```

cout << "Upgrading front" << endl;

for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    err = upgrade->startFrontUpgrade(path.c_str(), true);

```

```

if (CrossControl::ERR_SUCCESS == err) {
    cout << "Upgrade Ok" << endl;
    break;
}
else if(CrossControl::ERR_VERIFY_FAILED == err) {
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    err = upgrade->startFrontVerification(path.c_str(), true);

    if (CrossControl::ERR_SUCCESS == err) {
        cout << "Upgrade Ok" << endl;
        break;
    }
}
}
}

```

5.11.2.8 virtual eErr CrossControl::FirmwareUpgrade::startSSUpgrade (const char * filename, bool blocking) [pure virtual]

Start an upgrade of the System Supervisor microprocessor (SS). After an SS upgrade, the system must be shut down. The SS handles functions for shutting down of the computer. In order to shut down after an upgrade, shut down the OS and then toggle the power. The backlight will still be on after the OS has shut down.

Note that if you intend to do several upgrades/verifications in a row, the [Firmware-Upgrade](#) object should be released and reinitialised between each operation: `pFirmwareUpgrade->Release(); pFirmwareUpgrade = GetFirmwareUpgrade();`

Parameters

<i>filename</i>	Path and filename to the .hex file to program.
<i>blocking</i>	Whether or not the function should wait until completion. If blocking is set to false, the function will return immediately. One must then call <code>fpgaUpgradeStatus</code> to get the status of the upgrade operation. If blocking is set to true, the function will return when the operation is complete. This might take a few minutes.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

cout << "Upgrading SS" << endl;

for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);
}

```

```

err = upgrade->startSSUpgrade(path.c_str(), true);
if (CrossControl::ERR_SUCCESS == err) {
    cout << "Upgrade Ok" << endl;
    break;
}
else if(CrossControl::ERR_VERIFY_FAILED == err) {
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    err = upgrade->startSSVerification(path.c_str(), true);

    if (CrossControl::ERR_SUCCESS == err) {
        cout << "Upgrade Ok" << endl;
        break;
    }
}
}
}

```

5.11.2.9 virtual eErr CrossControl::FirmwareUpgrade::startSSVerification (const char * filename, bool blocking) [pure virtual]

Start a verification of the System Supervisor microprocessor (SS). Verifies the SS against the file to program. This could be useful if verification during programming fails.

Note that if you intend to do several upgrades/verifications in a row, the [FirmwareUpgrade](#) object should be released and reinitialised between each operation: pFirmwareUpgrade->[Release\(\)](#); pFirmwareUpgrade = [GetFirmwareUpgrade\(\)](#);

Parameters

<i>filename</i>	Path and filename to the .hex file to verify against.
<i>blocking</i>	Whether or not the function should wait until completion. If blocking is set to false, the function will return immediately. One must then call getUpgradeStatus to get the status of the operation. If blocking is set to true, the function will return when the operation is complete. This might take a few minutes.

Returns

error status. 0 = [ERR_SUCCESS](#), otherwise error code.

Example Usage:

```

cout << "Upgrading SS" << endl;

for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);
}

```

```

err = upgrade->startSSUpgrade(path.c_str(), true);
if (CrossControl::ERR_SUCCESS == err) {
    cout << "Upgrade Ok" << endl;
    break;
}
else if(CrossControl::ERR_VERIFY_FAILED == err) {
    // Reinitialize upgrade handle
    upgrade->Release();
    upgrade=GetFirmwareUpgrade();
    assert(upgrade != NULL);

    err = upgrade->startSSVerification(path.c_str(), true);

    if (CrossControl::ERR_SUCCESS == err) {
        cout << "Upgrade Ok" << endl;
        break;
    }
}
}
}

```

The documentation for this struct was generated from the following file:

- fixedIncludeFiles/[FirmwareUpgrade.h](#)

5.12 CrossControl::FrontLED Struct Reference

```
#include <FrontLED.h>
```

Public Member Functions

- virtual [eErr getSignal](#) (double *frequency, unsigned char *dutyCycle)=0
- virtual [eErr getOnTime](#) (unsigned char *onTime)=0
- virtual [eErr getOffTime](#) (unsigned char *offTime)=0
- virtual [eErr getIdleTime](#) (unsigned char *idleTime)=0
- virtual [eErr getNrOfPulses](#) (unsigned char *nrOfPulses)=0
- virtual [eErr getColor](#) (unsigned char *red, unsigned char *green, unsigned char *blue)=0
- virtual [eErr getColor](#) (CCAuxColor *color)=0
- virtual [eErr getEnabledDuringStartup](#) (CCStatus *status)=0
- virtual [eErr setSignal](#) (double frequency, unsigned char dutyCycle)=0
- virtual [eErr setOnTime](#) (unsigned char onTime)=0
- virtual [eErr setOffTime](#) (unsigned char offTime)=0
- virtual [eErr setIdleTime](#) (unsigned char idleTime)=0
- virtual [eErr setNrOfPulses](#) (unsigned char nrOfPulses)=0
- virtual [eErr setColor](#) (unsigned char red, unsigned char green, unsigned char blue)=0
- virtual [eErr setColor](#) (CCAuxColor color)=0
- virtual [eErr setOff](#) ()=0
- virtual [eErr setEnabledDuringStartup](#) (CCStatus status)=0
- virtual void [Release](#) ()=0

5.12.1 Detailed Description

Front LED control

Use the globally defined function [GetFrontLED\(\)](#) to get a handle to the [Diagnostic](#) struct. Use the method [FrontLED::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/frontled_example.cpp -lcc-aux -pthread -ldl */
#include <FrontLED.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>

using namespace std;
using namespace CrossControl;

void led_example(FRONTLEDHANDLE pFrontLED)
{
    CrossControl::eErr err;
    double freq;
    unsigned char dutycycle, red, green, blue;

    if(!pFrontLED)
        return;

    //
    // Get current settings for the LED
    //

    err = pFrontLED->getSignal(&freq, &dutycycle);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function getSignal: " << GetErrorStringA(
            err) << endl;
    }

    err = pFrontLED->getColor(&red, &green, &blue);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function getColor: " << GetErrorStringA(
            err) << endl;
    }

    //
    // Set LED blinking 2Hz red
    //

    cout << "Setting 2Hz red blink..." << endl;

    err = pFrontLED->setColor(RED);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function setColor: " << GetErrorStringA(
            err) << endl;
    }

    err = pFrontLED->setOffTime(25);
    if(err != ERR_SUCCESS)
    {
```

```

        cout << "Error(" << err << ") in function getColor: " << GetErrorStringA(
            err) << endl;
    }

    err = pFrontLED->setOnTime(25);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function getColor: " << GetErrorStringA(
            err) << endl;
    }

    cin.sync();
    cout << endl << "Press Enter restore the LED..." << endl;
    cin.get();

    //
    // Restore settings
    //

    err = pFrontLED->setSignal(freq, dutycycle);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function setSignal: " << GetErrorStringA(
            err) << endl;
    }

    err = pFrontLED->setColor(red, green, blue);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function setColor: " << GetErrorStringA(
            err) << endl;
    }
}

int main(void)
{
    FRONTLEDHANDLE pFrontLED = ::GetFrontLED();
    assert(pFrontLED);

    led_example(pFrontLED);

    pFrontLED->Release();
}

```

5.12.2 Member Function Documentation

5.12.2.1 virtual eErr CrossControl::FrontLED::getColor (unsigned char * *red*, unsigned char * *green*, unsigned char * *blue*) [pure virtual]

Get front LED color mix.

Parameters

<i>red</i>	Red color intensity 0-0x0F.
<i>green</i>	Green color intensity 0-0x0F.
<i>blue</i>	Blue color intensity 0-0x0F.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pFrontLED->getColor(&red, &green, &blue);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getColor: " << GetErrorStringA(
        err) << endl;
}
```

5.12.2.2 virtual eErr CrossControl::FrontLED::getColor (CCAuxColor * *color*) [pure virtual]

Get front LED color.

Parameters

<i>color</i>	Color from CCAuxColor
--------------	-----------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.3 virtual eErr CrossControl::FrontLED::getEnabledDuringStartup (CCStatus * *status*) [pure virtual]

Is the front LED enabled during startup? If enabled, the LED will blink yellow to indicate startup progress. It will turn green once the OS has started.

Parameters

<i>status</i>	LED Enabled or Disabled during startup.
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.4 virtual eErr CrossControl::FrontLED::getIdleTime (unsigned char * *idleTime*) [pure virtual]

Get front LED idle time.

Parameters

<i>idleTime</i>	Time in 100ms.
-----------------	----------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.5 `virtual eErr CrossControl::FrontLED::getNrOfPulses (unsigned char * nrOfPulses) [pure virtual]`

Get number of pulses during a blink sequence.

Parameters

<i>nrOfPulses</i>	Number of pulses.
-------------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.6 `virtual eErr CrossControl::FrontLED::getOffTime (unsigned char * offTime) [pure virtual]`

Get front LED off time.

Parameters

<i>offTime</i>	Time in 10ms.
----------------	---------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.7 `virtual eErr CrossControl::FrontLED::getOnTime (unsigned char * onTime) [pure virtual]`

Get front LED on time.

Parameters

<i>onTime</i>	Time in 10ms. 0 = off
---------------	-----------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

**5.12.2.8 virtual eErr CrossControl::FrontLED::getSignal (double * *frequency*,
 unsigned char * *dutyCycle*) [pure virtual]**

Get front LED signal. Note, the values may vary from previously set values with setSignal. This is due to precision-loss in approximations.

Parameters

<i>frequency</i>	LED blink frequency (0.2-50 Hz).
<i>dutyCycle</i>	LED on duty cycle (0-100%).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pFrontLED->getSignal(&freq, &dutycycle);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getSignal: " << GetErrorStringA(
        err) << endl;
}
```

5.12.2.9 virtual void CrossControl::FrontLED::Release () [pure virtual]

Delete the [FrontLED](#) object.

Returns

-

Example Usage:

```
FRONTLEDHANDLE pFrontLED = ::GetFrontLED();
assert(pFrontLED);

led_example(pFrontLED);

pFrontLED->Release();
```

**5.12.2.10 virtual eErr CrossControl::FrontLED::setColor (unsigned char *red*,
 unsigned char *green*, unsigned char *blue*) [pure virtual]**

Set front LED color mix.

Parameters

<i>red</i>	Red color intensity 0-0x0F.
<i>green</i>	Green color intensity 0-0x0F.
<i>blue</i>	Blue color intensity 0-0x0F.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pFrontLED->setColor(red, green, blue);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setColor: " << GetErrorStringA(
        err) << endl;
}
```

5.12.2.11 virtual eErr CrossControl::FrontLED::setColor (CCAuxColor *color*) [pure virtual]

Set one of the front LED standard colors.

Parameters

<i>color</i>	Color from CCAuxColor
--------------	-----------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pFrontLED->setColor(RED);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setColor: " << GetErrorStringA(
        err) << endl;
}
```

5.12.2.12 virtual eErr CrossControl::FrontLED::setEnabledDuringStartup (CCStatus *status*) [pure virtual]

Should the front LED be enabled during startup? If enabled, the LED will blink yellow to indicate startup progress. It will turn green once the OS has started.

Parameters

<i>status</i>	Enable or Disable the LED during startup.
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.13 `virtual eErr CrossControl::FrontLED::setIdleTime (unsigned char idleTime) [pure virtual]`

Get front LED idle time.

Parameters

<i>idleTime</i>	Time in 100ms.
-----------------	----------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.14 `virtual eErr CrossControl::FrontLED::setNrOfPulses (unsigned char nrOfPulses) [pure virtual]`

Set front LED number of pulses during a blink sequence.

Parameters

<i>nrOfPulses</i>	Number of pulses.
-------------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.15 `virtual eErr CrossControl::FrontLED::setOff() [pure virtual]`

Set front LED off.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.12.2.16 `virtual eErr CrossControl::FrontLED::setOffTime (unsigned char offTime) [pure virtual]`

Set front LED off time.

Parameters

<i>offTime</i>	Time in 10ms.
----------------	---------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pFrontLED->setOffTime(25);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getColor: " << GetErrorStringA(
        err) << endl;
}
```

5.12.2.17 virtual eErr CrossControl::FrontLED::setOnTime (unsigned char *onTime*)
 [pure virtual]

Set front LED on time.

Parameters

<i>onTime</i>	Time in 10ms. 0 = off
---------------	-----------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pFrontLED->setOnTime(25);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getColor: " << GetErrorStringA(
        err) << endl;
}
```

**5.12.2.18 virtual eErr CrossControl::FrontLED::setSignal (double *frequency*,
 unsigned char *dutyCycle*)** [pure virtual]

Set front LED signal.

Parameters

<i>frequency</i>	LED blink frequency (0.2-50 Hz).
<i>dutyCycle</i>	LED on duty cycle (0-100%).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:


```
err = pFrontLED->setSignal(freq, dutycycle);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setSignal: " << GetErrorStringA(
        err) << endl;
}
```

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/FrontLED.h](#)

5.13 CrossControl::LedColorMixType Struct Reference

```
#include <CCAuxTypes.h>
```

Public Attributes

- unsigned char [red](#)
- unsigned char [green](#)
- unsigned char [blue](#)

5.13.1 Member Data Documentation

5.13.1.1 unsigned char CrossControl::LedColorMixType::blue

Blue color intensity 0-0x0F

5.13.1.2 unsigned char CrossControl::LedColorMixType::green

Green color intensity 0-0x0F

5.13.1.3 unsigned char CrossControl::LedColorMixType::red

Red color intensity 0-0x0F

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/CCAuxTypes.h](#)

5.14 CrossControl::LedTimingType Struct Reference

```
#include <CCAuxTypes.h>
```

Public Attributes

- unsigned char [onTime](#)
- unsigned char [offTime](#)
- unsigned char [idleTime](#)
- unsigned char [nrOfPulses](#)

5.14.1 Member Data Documentation**5.14.1.1 unsigned char CrossControl::LedTimingType::idleTime**

LED idle time in 100ms

5.14.1.2 unsigned char CrossControl::LedTimingType::nrOfPulses

Pulses per sequences

5.14.1.3 unsigned char CrossControl::LedTimingType::offTime

LED off time in 10ms

5.14.1.4 unsigned char CrossControl::LedTimingType::onTime

LED on time in 10ms

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/CCAuxTypes.h](#)

5.15 CrossControl::Lightsensor Struct Reference

```
#include <Lightsensor.h>
```

Public Member Functions

- virtual [eErr getIlluminance](#) (unsigned short *value)=0
- virtual [eErr getIlluminance](#) (unsigned short *value, unsigned char *ch0, unsigned char *ch1)=0
- virtual [eErr getAverageIlluminance](#) (unsigned short *value)=0
- virtual [eErr startAverageCalc](#) (unsigned long averageWndSize, unsigned long rejectWndSize, unsigned long rejectDeltaInLux, [LightSensorSamplingMode](#) mode)=0
- virtual [eErr stopAverageCalc](#) ()=0
- virtual [eErr getOperatingRange](#) ([LightSensorOperationRange](#) *range)=0
- virtual [eErr setOperatingRange](#) ([LightSensorOperationRange](#) range)=0
- virtual void [Release](#) ()=0

5.15.1 Detailed Description

Light Sensor access. Note that reading the light sensor using average calculation at the same time as running the automatic backlight control is not supported. Also note that Lux values mentioned below (and in the [Backlight](#) class) are not necessarily true lux values. The values received are lower than true lux values, due to the light guide in the front panel, where some light is lost. It is still a measurement of the illuminance (in Lux).

Use the globally defined function [GetLightsensor\(\)](#) to get a handle to the [Lightsensor](#) struct. Use the method [Lightsensor::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/lightsensor_example.cpp -lcc-aux -pthread -ldl */
#include <Lightsensor.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>

using namespace std;
using namespace CrossControl;

void ls_example(LIGHTSENSORHANDLE pLightSensor)
{
    CrossControl::eErrr err;
    unsigned short value;

    err = pLightSensor->getIlluminance(&value);
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function getIlluminance: " <<
            GetErrorStringA(err) << endl;
    }

    cout << "Lightsensor value: " << (int)value << endl;

    cout << "Starting average calculation..." << endl;

    // Start the average calculation background function
    // This cannot be used if the automatic backlihg function is running.
    err = pLightSensor->startAverageCalc(5, 5, 50, SamplingModeAuto);
    if(err == ERR_AVERAGE_CALC_STARTED)
    {
        cout << "Error(" << err << ") in function startAverageCalc: " <<
            GetErrorStringA(err) << endl;
        cout << endl << "Please turn off Automatic backlight! (CCsettings - Display
            tab)" << endl;
        return;
    }
    else if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function startAverageCalc: " <<
            GetErrorStringA(err) << endl;
    }

    cin.sync();
    cout << endl << "Press Enter to read the average value..." << endl;
    cin.get();

    err = pLightSensor->getAverageIlluminance(&value);
```

```

if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getAverageIlluminance: " <<
        GetErrorStringA(err) << endl;
}

cout << "Lightsensor average value: " << (int)value << endl;

err = pLightSensor->stopAverageCalc();
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function stopAverageCalc: " <<
        GetErrorStringA(err) << endl;
}
}

int main(void)
{
    LIGHTSENSORHANDLE pLightSensor = ::GetLightsensor();
    assert(pLightSensor);

    ls_example(pLightSensor);

    pLightSensor->Release();
}

```

5.15.2 Member Function Documentation

5.15.2.1 virtual eErr CrossControl::Lightsensor::getAverageIlluminance (unsigned short * *value*) [pure virtual]

Get average illuminance (light) value from light sensor.

Parameters

<i>value</i>	Illuminance value (Lux).
--------------	--------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pLightSensor->getAverageIlluminance(&value);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getAverageIlluminance: " <<
        GetErrorStringA(err) << endl;
}

```

5.15.2.2 virtual eErr CrossControl::Lightsensor::getIlluminance (unsigned short * *value*) [pure virtual]

Get illuminance (light) value from light sensor.

Parameters

<i>value</i>	Illuminace value (Lux).
--------------	-------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pLightSensor->getIlluminance(&value);
if (err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function getIlluminance: " <<
        GetErrorStringA(err) << endl;
}
```

5.15.2.3 virtual eErr CrossControl::Lightsensor::getIlluminance (unsigned short * *value*, unsigned char * *ch0*, unsigned char * *ch1*) [pure virtual]

Get illuminance (light) value from light sensor.

Parameters

<i>value</i>	Illuminance value (Lux).
<i>ch0</i>	Channel0 value.
<i>ch1</i>	Channel1 value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.15.2.4 virtual eErr CrossControl::Lightsensor::getOperatingRange (LightSensorOperationRange * *range*) [pure virtual]

Get operating range. The light sensor can operate in two ranges. Standard and extended range. In standard range, the range is smaller but resolution higher. See the TSL2550 data sheet for more information.

Parameters

<i>range</i>	Operating range. RangeStandard or RangeExtended.
--------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.15.2.5 virtual void CrossControl::Lightsensor::Release () [pure virtual]

Delete the [Lightsensor](#) object.

Returns

-

Example Usage:

```

LIGHTSENSORHANDLE pLightSensor = ::GetLightsensor();
assert (pLightSensor);

ls_example (pLightSensor);

pLightSensor->Release ();

```

5.15.2.6 virtual eErr CrossControl::Lightsensor::setOperatingRange (LightsensorOperationRange *range*) [pure virtual]

Set operating range. The light sensor can operate in two ranges. Standard and extended range. In standard range, the range is smaller but resolution higher. See the TSL2550 data sheet for more information.

Parameters

<i>range</i>	Operating range to set. RangeStandard or RangeExtended.
--------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.15.2.7 virtual eErr CrossControl::Lightsensor::startAverageCalc (unsigned long *averageWndSize*, unsigned long *rejectWndSize*, unsigned long *rejectDeltaInLux*, LightsensorSamplingMode *mode*) [pure virtual]

Start average calculation.

Parameters

<i>average-WndSize</i>	The average window size in nr of samples.
<i>rejectWnd-Size</i>	The reject window size in nr of samples.
<i>rejectDelta-InLux</i>	The reject delta in lux.
<i>mode</i>	The configured sampling mode.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
// Start the average calculation background function
// This cannot be used if the automatic backlight function is running.
err = pLightSensor->startAverageCalc(5, 5, 50, SamplingModeAuto);
if(err == ERR_AVERAGE_CALC_STARTED)
{
    cout << "Error(" << err << ") in function startAverageCalc: " <<
        GetErrorStringA(err) << endl;
    cout << endl << "Please turn off Automatic backlight! (CCsettings - Display
        tab)" << endl;
    return;
}
else if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function startAverageCalc: " <<
        GetErrorStringA(err) << endl;
}
```

5.15.2.8 virtual eErr CrossControl::Lightsensor::stopAverageCalc () [pure virtual]

Stop average calculation.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pLightSensor->stopAverageCalc();
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function stopAverageCalc: " <<
        GetErrorStringA(err) << endl;
}
```

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Lightsensor.h](#)

5.16 CrossControl::Power Struct Reference

```
#include <Power.h>
```

Public Member Functions

- virtual eErr [getBLPowerStatus](#) (CCStatus *status)=0

- virtual `eErr getCanPowerStatus (CCStatus *status)=0`
- virtual `eErr getVideoPowerStatus (unsigned char *videoStatus)=0`
- virtual `eErr getExtFanPowerStatus (CCStatus *status)=0`
- virtual `eErr setBLPowerStatus (CCStatus status)=0`
- virtual `eErr setCanPowerStatus (CCStatus status)=0`
- virtual `eErr setVideoPowerStatus (unsigned char status)=0`
- virtual `eErr setExtFanPowerStatus (CCStatus status)=0`
- virtual `eErr getButtonPowerTransitionStatus (ButtonPowerTransitionStatus *status)=0`
- virtual `eErr ackPowerRequest ()=0`
- virtual void `Release ()=0`

5.16.1 Detailed Description

Power control access functions

Use the globally defined function `GetPower()` to get a handle to the `Power` struct. Use the method `Power::Release()` to return the handle.

Example Usage:

```

/* g++ -DLINUX examples/power_example.cpp -lcc-aux -pthread -ldl */
#include <Power.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>

using namespace std;
using namespace CrossControl;

void power_example(POWERHANDLE pPower)
{
    CrossControl::eErr err;
    unsigned char value;
    CCStatus status;

    err = pPower->getBLPowerStatus(&status);
    if(err == ERR_SUCCESS)
    {
        cout << "Backlight power is " << ((status == Enabled)? "ON" : "OFF") <<
            endl;
    }
    else
    {
        cout << "Error(" << err << ") in function getBLPowerStatus: " <<
            GetErrorStringA(err) << endl;
    }

    err = pPower->getVideoPowerStatus(&value);
    if(err == ERR_SUCCESS)
    {
        cout << "Video power status: " << endl;
        cout << "Video1: " << ((value & 0x01)? "ON" : "OFF") << endl;
        cout << "Video2: " << ((value & 0x02)? "ON" : "OFF") << endl;
        cout << "Video3: " << ((value & 0x04)? "ON" : "OFF") << endl;
        cout << "Video4: " << ((value & 0x08)? "ON" : "OFF") << endl;
    }
}

```



```

else
{
    cout << "Error(" << err << ") in function getVideoPowerStatus: " <<
        GetErrorStringA(err) << endl;
}

cout << "Blinking backlight... " << endl;
cin.sync();
cout << endl << "Press Enter to to turn off the Backlight and then Enter to
    turn it on again..." << endl;
cin.get();
err = pPower->setBLPowerStatus(Disabled);
cin.sync();
cin.get();
err = pPower->setBLPowerStatus(Enabled);
if(err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setBLPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
}

int main(void)
{
    POWERHANDLE pPower = ::GetPower();
    assert(pPower);

    power_example(pPower);

    pPower->Release();
}

```

5.16.2 Member Function Documentation

5.16.2.1 virtual eErr CrossControl::Power::ackPowerRequest() [pure virtual]

Acknowledge a power request from the system supervisor. This is handled by the service/daemon and should normally not be used by applications unless the [Cross-Control](#) service/daemon is not being run on the system. If that is the case, the following requests (read by `getButtonPowerTransitionStatus`) should be acknowledged: `BPTS_ShutDown`, `BPTS_Suspend` and `BPTS_Restart`

Returns

error status. 0 = `ERR_SUCCESS`, otherwise error code.

5.16.2.2 virtual eErr CrossControl::Power::getBLPowerStatus(CCStatus * status) [pure virtual]

Get backlight power status.

Parameters

<i>status</i>	Backlight power status.
---------------	-------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pPower->getBLPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "Backlight power is " << ((status == Enabled)? "ON" : "OFF") <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getBLPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

5.16.2.3 virtual eErr CrossControl::Power::getButtonPowerTransitionStatus (ButtonPowerTransitionStatus * *status*) [pure virtual]

Get the current status for front panel button and on/off signal.

Parameters

<i>status</i>	The current status. See the definition of ButtonPowerTransitionStatus for details.
---------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.16.2.4 virtual eErr CrossControl::Power::getCanPowerStatus (CCStatus * *status*) [pure virtual]

Get can power status.

Parameters

<i>status</i>	Can power status.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.16.2.5 `virtual eErr CrossControl::Power::getExtFanPowerStatus (CCStatus * status)` [pure virtual]

Get external fan power status.

Parameters

<i>status</i>	Fan power status.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.16.2.6 `virtual eErr CrossControl::Power::getVideoPowerStatus (unsigned char * videoStatus)` [pure virtual]

Get [Video](#) power status.

Parameters

<i>videoStatus</i>	Video power status. Bit0: Video 1. Bit1: Video 2. Bit2: Video 3. Bit3: Video 4. (1=on, 0=off)
--------------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pPower->getVideoPowerStatus(&value);
if(err == ERR_SUCCESS)
{
    cout << "Video power status: " << endl;
    cout << "Video1: " << ((value & 0x01)? "ON" : "OFF") << endl;
    cout << "Video2: " << ((value & 0x02)? "ON" : "OFF") << endl;
    cout << "Video3: " << ((value & 0x04)? "ON" : "OFF") << endl;
    cout << "Video4: " << ((value & 0x08)? "ON" : "OFF") << endl;
}
else
{
    cout << "Error(" << err << ") in function getVideoPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

5.16.2.7 `virtual void CrossControl::Power::Release ()` [pure virtual]

Delete the [Power](#) object.

Returns

-

Example Usage:

```
POWERHANDLE pPower = ::GetPower();
assert (pPower);

power_example (pPower);

pPower->Release();
```

5.16.2.8 virtual eErr CrossControl::Power::setBLPowerStatus (CCStatus *status*)
 [pure virtual]

Set backlight power status.

Parameters

<i>status</i>	Backlight power status.
---------------	-------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
cout << "Blinking backlight... " << endl;
cin.sync();
cout << endl << "Press Enter to to turn off the Backlight and then Enter to
    turn it on again..." << endl;
cin.get();
err = pPower->setBLPowerStatus(Disabled);
cin.sync();
cin.get();
err = pPower->setBLPowerStatus(Enabled);
if (err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setBLPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

5.16.2.9 virtual eErr CrossControl::Power::setCanPowerStatus (CCStatus *status*)
 [pure virtual]

Set can power status.

Parameters

<i>status</i>	Can power status.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.16.2.10 `virtual eErr CrossControl::Power::setExtFanPowerStatus (CCStatus status) [pure virtual]`

Set external fan power status.

Parameters

<i>status</i>	Fan power status.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.16.2.11 `virtual eErr CrossControl::Power::setVideoPowerStatus (unsigned char status) [pure virtual]`

Set [Video](#) power status.

Parameters

<i>status</i>	Video power status. Bit0: Video 1. Bit1: Video 2. Bit2: Video 3. Bit3: Video 4. (1=on, 0=off)
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Power.h](#)

5.17 CrossControl::received_video Struct Reference

```
#include <CCAuxTypes.h>
```

Public Attributes

- unsigned short [received_width](#)
- unsigned short [received_height](#)
- unsigned char [received_framerate](#)

5.17.1 Member Data Documentation

5.17.1.1 unsigned char CrossControl::received_video::received_framerate

5.17.1.2 unsigned short CrossControl::received_video::received_height

5.17.1.3 unsigned short CrossControl::received_video::received_width

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/CCAuxTypes.h](#)

5.18 CrossControl::Telematics Struct Reference

```
#include <Telematics.h>
```

Public Member Functions

- virtual [eErr](#) [getTelematicsAvailable](#) (CCStatus *status)=0
- virtual [eErr](#) [getGPRSPowerStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [getGPRSStartupPowerStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [getWLANPowerStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [getWLANStartupPowerStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [getGPSAntennaStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [setGPRSPowerStatus](#) (CCStatus status)=0
- virtual [eErr](#) [setGPRSStartupPowerStatus](#) (CCStatus status)=0
- virtual [eErr](#) [setWLANPowerStatus](#) (CCStatus status)=0
- virtual [eErr](#) [setWLANStartupPowerStatus](#) (CCStatus status)=0
- virtual void [Release](#) ()=0
- virtual [eErr](#) [getBTPowerStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [getBTStartupPowerStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [getGPSPowerStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [getGPSStartupPowerStatus](#) (CCStatus *status)=0
- virtual [eErr](#) [setBTPowerStatus](#) (CCStatus status)=0
- virtual [eErr](#) [setBTStartupPowerStatus](#) (CCStatus status)=0
- virtual [eErr](#) [setGPSPowerStatus](#) (CCStatus status)=0
- virtual [eErr](#) [setGPSStartupPowerStatus](#) (CCStatus status)=0

5.18.1 Detailed Description

Power control and status functions for the optional telematics add-on card

Use the globally defined function [GetTelematics\(\)](#) to get a handle to the [Telematics](#) struct. Use the method [Telematics::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/telematics_example.cpp -lcc-aux -pthread -ldl */
#include <Telematics.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>

using namespace std;
using namespace CrossControl;

void telematics_example(TELEMATICSHANDLE pTelematics)
{
    CrossControl::eErrr err;
    CCStatus status;

    err = pTelematics->getTelematicsAvailable(&status);
    if(err == ERR_SUCCESS)
    {
        cout << "Telematics add-on board: " << ((status == Enabled)? "available" :
            "not available") << endl;
        if(status == Disabled)
            return;
    }
    else
    {
        cout << "Error(" << err << ") in function getTelematicsAvailable: " <<
            GetErrorStringA(err) << endl;
        return;
    }

    err = pTelematics->getBTPowerStatus(&status);
    if(err == ERR_SUCCESS)
    {
        cout << "Bluetooth power is " << ((status == Enabled)? "ON" : "OFF") <<
            endl;
    }
    else if(err == ERR_TELEMATICS_BT_NOT_AVAILABLE)
    {
        cout << "getBLPowerStatus: Bluetooth is not available on this platform" <<
            endl;
    }
    else
    {
        cout << "Error(" << err << ") in function getBLPowerStatus: " <<
            GetErrorStringA(err) << endl;
    }

    err = pTelematics->getBTStartUpPowerStatus(&status);
    if(err == ERR_SUCCESS)
    {
        cout << "Bluetooth power is " << ((status == Enabled)? "Enabled" : "
            Disabled") << " at start-up" << endl;
    }
    else if(err == ERR_TELEMATICS_BT_NOT_AVAILABLE)
    {
        cout << "getBTStartUpPowerStatus: Bluetooth is not available on this
            platform" << endl;
    }
    else
    {
        cout << "Error(" << err << ") in function getBTStartUpPowerStatus: " <<
            GetErrorStringA(err) << endl;
    }
}
```

```
}

err = pTelematics->getGPRSPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GSM/GPRS power is " << ((status == Enabled)? "ON" : "OFF") << endl
        ;
}
else if(err == ERR_TELEMATICS_GPRS_NOT_AVAILABLE)
{
    cout << "getGPRSPowerStatus: GSM/GPRS is not available on this platform" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getGPRSPowerStatus: " <<
        GetErrorStringA(err) << endl;
}

err = pTelematics->getGPRSStartUpPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GSM/GPRS power is " << ((status == Enabled)? "Enabled" : "Disabled
        ") << " at start-up" << endl;
}
else if(err == ERR_TELEMATICS_GPRS_NOT_AVAILABLE)
{
    cout << "getGPRSStartUpPowerStatus: GSM/GPRS is not available on this
        platform" << endl;
}
else
{
    cout << "Error(" << err << ") in function getGPRSStartUpPowerStatus: " <<
        GetErrorStringA(err) << endl;
}

err = pTelematics->getGPSPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GPS power is " << ((status == Enabled)? "ON" : "OFF") << endl;
}
else if(err == ERR_TELEMATICS_GPS_NOT_AVAILABLE)
{
    cout << "getGPSPowerStatus: GPS is not available on this platform" << endl;
}
else
{
    cout << "Error(" << err << ") in function getGPSPowerStatus: " <<
        GetErrorStringA(err) << endl;
}

err = pTelematics->getGPSStartUpPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GPS power is " << ((status == Enabled)? "Enabled" : "Disabled") <<
        " at start-up" << endl;
}
else if(err == ERR_TELEMATICS_GPS_NOT_AVAILABLE)
{
    cout << "getGPSStartUpPowerStatus: GPS is not available on this platform" <
```



```

        < endl;
    }
    else
    {
        cout << "Error(" << err << ") in function getGPSStartUpPowerStatus: " <<
            GetErrorStringA(err) << endl;
    }

    err = pTelematics->getGPSAntennaStatus(&status);
    if(err == ERR_SUCCESS)
    {
        cout << "GPS antenna status: " << ((status == Enabled)? "OK" : "ERROR: Open
            connection or short-circuit") << endl;
    }
    else if(err == ERR_TELEMATICS_GPS_NOT_AVAILABLE)
    {
        cout << "getGPSAntennaStatus: GPS is not available on this platform" <<
            endl;
    }
    else
    {
        cout << "Error(" << err << ") in function getGPSAntennaStatus: " <<
            GetErrorStringA(err) << endl;
    }
}

err = pTelematics->getWLANPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "WLAN power is " << ((status == Enabled)? "ON" : "OFF") << endl;
}
else if(err == ERR_TELEMATICS_WLAN_NOT_AVAILABLE)
{
    cout << "getWLANPowerStatus: WLAN is not available on this platform" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getWLANPowerStatus: " <<
        GetErrorStringA(err) << endl;
}

err = pTelematics->getWLANStartUpPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "WLAN power is " << ((status == Enabled)? "Enabled" : "Disabled") <
        < " at start-up" << endl;
}
else if(err == ERR_TELEMATICS_WLAN_NOT_AVAILABLE)
{
    cout << "getWLANStartUpPowerStatus: WLAN is not available on this platform"
        << endl;
}
else
{
    cout << "Error(" << err << ") in function getWLANStartUpPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
}

int main(void)
{

```

```

TELEMATICSHANDLE pTelematics = ::GetTelematics();
assert(pTelematics);

telematics_example(pTelematics);

pTelematics->Release();
}

```

5.18.2 Member Function Documentation

5.18.2.1 virtual eErr CrossControl::Telematics::getBTPowerStatus (CCStatus * status) [pure virtual]

Get Bluetooth power status.

Parameters

<i>status</i>	Bluetooth power status.
---------------	-------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pTelematics->getBTPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "Bluetooth power is " << ((status == Enabled)? "ON" : "OFF") <<
        endl;
}
else if(err == ERR_TELEMATICS_BT_NOT_AVAILABLE)
{
    cout << "getBTPowerStatus: Bluetooth is not available on this platform" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getBTPowerStatus: " <<
        GetErrorStringA(err) << endl;
}

```

5.18.2.2 virtual eErr CrossControl::Telematics::getBTStartupPowerStatus (CCStatus * status) [pure virtual]

Get Bluetooth power status at startup and at resume from suspended mode.

Parameters

<i>status</i>	Bluetooth power status.
---------------	-------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTelematics->getBTStartUpPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "Bluetooth power is " << ((status == Enabled)? "Enabled" : "
        Disabled") << " at start-up" << endl;
}
else if(err == ERR_TELEMATICS_BT_NOT_AVAILABLE)
{
    cout << "getBTStartUpPowerStatus: Bluetooth is not available on this
        platform" << endl;
}
else
{
    cout << "Error(" << err << ") in function getBTStartUpPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

5.18.2.3 virtual eErr CrossControl::Telematics::getGPRSPowerStatus (CCStatus * *status*) [pure virtual]

Get GPRS power status.

Parameters

<i>status</i>	GPRS power status.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTelematics->getGPRSPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GSM/GPRS power is " << ((status == Enabled)? "ON" : "OFF") << endl
        ;
}
else if(err == ERR_TELEMATICS_GPRS_NOT_AVAILABLE)
{
    cout << "getGPRSPowerStatus: GSM/GPRS is not available on this platform" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getGPRSPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

5.18.2.4 virtual eErr CrossControl::Telematics::getGPRSStartupPowerStatus (CCStatus * status) [pure virtual]

Get GPRS power status at startup and at resume from suspended mode.

Parameters

<i>status</i>	GPRS power status.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTelematics->getGPRSStartupPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GSM/GPRS power is " << ((status == Enabled)? "Enabled" : "Disabled
    ") << " at start-up" << endl;
}
else if(err == ERR_TELEMATICS_GPRS_NOT_AVAILABLE)
{
    cout << "getGPRSStartupPowerStatus: GSM/GPRS is not available on this
    platform" << endl;
}
else
{
    cout << "Error(" << err << ") in function getGPRSStartupPowerStatus: " <<
    GetErrorStringA(err) << endl;
}
```

5.18.2.5 virtual eErr CrossControl::Telematics::getGPSAntennaStatus (CCStatus * status) [pure virtual]

Get GPS antenna status. Antenna open/short detection. The status is set to disabled if no antenna is present or a short is detected.

Parameters

<i>status</i>	GPS antenna power status.
---------------	---------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTelematics->getGPSAntennaStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GPS antenna status: " << ((status == Enabled)? "OK" : "ERROR: Open
    connection or short-circuit") << endl;
}
```

```

}
else if (err == ERR_TELEMATICS_GPS_NOT_AVAILABLE)
{
    cout << "getGPSAntennaStatus: GPS is not available on this platform" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getGPSAntennaStatus: " <<
        GetErrorStringA(err) << endl;
}

```

5.18.2.6 virtual eErr CrossControl::Telematics::getGPSPowerStatus (CCStatus * status) [pure virtual]

Get GPS power status. Note that it can take some time after calling setGPSPowerStatus before the status is reported correctly.

Parameters

<i>status</i>	GPS power status.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pTelematics->getGPSPowerStatus(&status);
if (err == ERR_SUCCESS)
{
    cout << "GPS power is " << ((status == Enabled)? "ON" : "OFF") << endl;
}
else if (err == ERR_TELEMATICS_GPS_NOT_AVAILABLE)
{
    cout << "getGPSPowerStatus: GPS is not available on this platform" << endl;
}
else
{
    cout << "Error(" << err << ") in function getGPSPowerStatus: " <<
        GetErrorStringA(err) << endl;
}

```

5.18.2.7 virtual eErr CrossControl::Telematics::getGPSStartupPowerStatus (CCStatus * status) [pure virtual]

Get GPS power status at startup and at resume from suspended mode.

Parameters

<i>status</i>	GPS power status.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTelematics->getGPSStartUpPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GPS power is " << ((status == Enabled)? "Enabled" : "Disabled") <<
        " at start-up" << endl;
}
else if(err == ERR_TELEMATICS_GPS_NOT_AVAILABLE)
{
    cout << "getGPSStartUpPowerStatus: GPS is not available on this platform" <
        < endl;
}
else
{
    cout << "Error(" << err << ") in function getGPSStartUpPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

5.18.2.8 virtual eErr CrossControl::Telematics::getTelematicsAvailable (CCStatus * *status*) [pure virtual]

Is a telematics add-on card installed?

Parameters

<i>status</i>	Enabled if a telematics add-on card is installed, otherwise Disabled.
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTelematics->getTelematicsAvailable(&status);
if(err == ERR_SUCCESS)
{
    cout << "Telematics add-on board: " << ((status == Enabled)? "available" :
        "not available") << endl;
    if(status == Disabled)
        return;
}
else
{
    cout << "Error(" << err << ") in function getTelematicsAvailable: " <<
        GetErrorStringA(err) << endl;
    return;
}
```

5.18.2.9 virtual eErr CrossControl::Telematics::getWLANPowerStatus (CCStatus * *status*) [pure virtual]

Get WLAN power status.

Parameters

<i>status</i>	WLAN power status.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTelematics->getWLANPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "WLAN power is " << ((status == Enabled)? "ON" : "OFF") << endl;
}
else if(err == ERR_TELEMATICS_WLAN_NOT_AVAILABLE)
{
    cout << "getWLANPowerStatus: WLAN is not available on this platform" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getWLANPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

5.18.2.10 virtual eErr CrossControl::Telematics::getWLANStartUpPowerStatus (CCStatus * *status*) [pure virtual]

Get WLAN power status at startup and at resume from suspended mode.

Parameters

<i>status</i>	WLAN power status.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTelematics->getWLANStartUpPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "WLAN power is " << ((status == Enabled)? "Enabled" : "Disabled") <
        < " at start-up" << endl;
}
else if(err == ERR_TELEMATICS_WLAN_NOT_AVAILABLE)
```

```

{
    cout << "getWLANStartUpPowerStatus: WLAN is not available on this platform"
        << endl;
}
else
{
    cout << "Error(" << err << ") in function getWLANStartUpPowerStatus: " <<
        GetErrorStringA(err) << endl;
}

```

5.18.2.11 virtual void CrossControl::Telematics::Release() [pure virtual]

Delete the [Telematics](#) object.

Returns

-

Example Usage:

```

TELEMATICSHANDLE pTelematics = ::GetTelematics();
assert(pTelematics);

telematics_example(pTelematics);

pTelematics->Release();

```

5.18.2.12 virtual eErr CrossControl::Telematics::setBTPowerStatus (CCStatus *status*) [pure virtual]

Set Bluetooth power status.

Parameters

<i>status</i>	Bluetooth power status.
---------------	-------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.18.2.13 virtual eErr CrossControl::Telematics::setBTStartUpPowerStatus (CCStatus *status*) [pure virtual]

Set Bluetooth power status at startup and at resume from suspended mode.

Parameters

<i>status</i>	Bluetooth power status.
---------------	-------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.18.2.14 `virtual eErr CrossControl::Telematics::setGPRSPowerStatus (CCStatus status) [pure virtual]`

Set GPRS modem power status.

Parameters

<i>status</i>	GPRS modem power status.
---------------	--------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.18.2.15 `virtual eErr CrossControl::Telematics::setGPRSStartupPowerStatus (CCStatus status) [pure virtual]`

Set GPRS power status at startup and at resume from suspended mode.

Parameters

<i>status</i>	GPRS power status.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.18.2.16 `virtual eErr CrossControl::Telematics::setGPSPowerStatus (CCStatus status) [pure virtual]`

Set GPS power status.

Parameters

<i>status</i>	GPS power status.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.18.2.17 virtual eErr CrossControl::Telematics::setGPSStartUpPowerStatus (CCStatus *status*) [pure virtual]

Set GPS power status at startup and at resume from suspended mode.

Parameters

<i>status</i>	GPS power status.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.18.2.18 virtual eErr CrossControl::Telematics::setWLANPowerStatus (CCStatus *status*) [pure virtual]

Set WLAN power status.

Parameters

<i>status</i>	WLAN power status.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.18.2.19 virtual eErr CrossControl::Telematics::setWLANStartUpPowerStatus (CCStatus *status*) [pure virtual]

Set WLAN power status at startup and at resume from suspended mode.

Parameters

<i>status</i>	WLAN power status.
---------------	--------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Telematics.h](#)

5.19 CrossControl::TimerType Struct Reference

```
#include <CCAuxTypes.h>
```

Public Attributes

- unsigned long [TotRunTime](#)
- unsigned long [TotSuspTime](#)
- unsigned long [TotHeatTime](#)
- unsigned long [RunTime40_60](#)
- unsigned long [RunTime60_70](#)
- unsigned long [RunTime70_80](#)
- unsigned long [Above80RunTime](#)

5.19.1 Detailed Description

[Diagnostic](#) timer data

5.19.2 Member Data Documentation

5.19.2.1 unsigned long CrossControl::TimerType::Above80RunTime

Total runtime in 70-80deg (minutes)

5.19.2.2 unsigned long CrossControl::TimerType::RunTime40_60

Total heating time (minutes)

5.19.2.3 unsigned long CrossControl::TimerType::RunTime60_70

Total runtime in 40-60deg (minutes)

5.19.2.4 unsigned long CrossControl::TimerType::RunTime70_80

Total runtime in 60-70deg (minutes)

5.19.2.5 unsigned long CrossControl::TimerType::TotHeatTime

Total suspend time (minutes)

5.19.2.6 unsigned long CrossControl::TimerType::TotRunTime

5.19.2.7 unsigned long CrossControl::TimerType::TotSuspTime

Total running time (minutes)

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/CCAuxTypes.h](#)

5.20 CrossControl::TouchScreen Struct Reference

```
#include <TouchScreen.h>
```

Public Member Functions

- virtual [eErr](#) [getMode](#) ([TouchScreenModeSettings](#) *config)=0
- virtual [eErr](#) [getMouseRightClickTime](#) (unsigned short *time)=0
- virtual [eErr](#) [setMode](#) ([TouchScreenModeSettings](#) config)=0
- virtual [eErr](#) [setMouseRightClickTime](#) (unsigned short time)=0
- virtual [eErr](#) [setAdvancedSetting](#) ([TSAdvancedSettingsParameter](#) param, unsigned short data)=0
- virtual [eErr](#) [getAdvancedSetting](#) ([TSAdvancedSettingsParameter](#) param, unsigned short *data)=0
- virtual void [Release](#) ()=0

5.20.1 Detailed Description

Touch Screen settings

Use the globally defined function [GetTouchScreen\(\)](#) to get a handle to the [TouchScreen](#) struct. Use the method [TouchScreen::Release\(\)](#) to return the handle.

Example Usage:

```
/* g++ -DLINUX examples/touchscreen_example.cpp -lcc-aux -pthread -ldl */
#include <TouchScreen.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>

using namespace std;
using namespace CrossControl;

void touchscreen_example(TOUCHSCREENHANDLE pTouchScreen)
{
    CrossControl::eErr err;
    unsigned short rightclicktime, debouncetime;
    TouchScreenModeSettings ts_mode;

    err = pTouchScreen->getMode(&ts_mode);
```

```
if(err == ERR_SUCCESS)
{
    switch(ts_mode)
    {
        case MOUSE_NEXT_BOOT: cout << "USB profile is set to Mouse profile (active
            next boot)" << endl; break;
        case TOUCH_NEXT_BOOT: cout << "USB profile is set to Touch profile (active
            next boot)" << endl; break;
        case MOUSE_NOW: cout << "USB profile is set to Mouse profile" << endl;
            break;
        case TOUCH_NOW: cout << "USB profile is set to Touch profile" << endl;
            break;
        default: cout << "Error: invalid setting returned from getMode" << endl;
            break;
    }
}
else
{
    cout << "Error(" << err << ") in function getMode: " << GetErrorStringA(err
        ) << endl;
}

err = pTouchScreen->getMouseRightClickTime(&rightclicktime);
if(err == ERR_SUCCESS)
{
    cout << "Right click time is set to: " << (int)rightclicktime << " ms" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getMouseRightClickTime: " <<
        GetErrorStringA(err) << endl;
}

err = pTouchScreen->getAdvancedSetting(TS_DEBOUNCE_TIME, &debouncetime);
if(err == ERR_SUCCESS)
{
    cout << "Touchscreen debounce time is set to: " << (int)debouncetime << "
        ms" << endl;
}
else
{
    cout << "Error(" << err << ") in function getAdvancedSetting: " <<
        GetErrorStringA(err) << endl;
}
}

int main(void)
{
    TOUCHSCREENHANDLE pTouchScreen = ::GetTouchScreen();
    assert(pTouchScreen);

    touchscreen_example(pTouchScreen);

    pTouchScreen->Release();
}
```

5.20.2 Member Function Documentation

5.20.2.1 virtual eErr CrossControl::TouchScreen::getAdvancedSetting (TSAdvancedSettingsParameter *param*, unsigned short * *data*) [pure virtual]

Get advanced touch screen settings. See the description of TSAdvancedSettingsParameter for a description of the parameters.

Parameters

<i>param</i>	The setting to get.
<i>data</i>	The current data for the setting.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTouchScreen->getAdvancedSetting(TS_DEBOUNCE_TIME, &debouncetime);
if(err == ERR_SUCCESS)
{
    cout << "Touchscreen debounce time is set to: " << (int)debouncetime << "
        ms" << endl;
}
else
{
    cout << "Error(" << err << ") in function getAdvancedSetting: " <<
        GetErrorStringA(err) << endl;
}
```

5.20.2.2 virtual eErr CrossControl::TouchScreen::getMode (TouchScreenModeSettings * *config*) [pure virtual]

Get Touch Screen mode. Gets the current mode of the USB profile.

Parameters

<i>config</i>	The current mode.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTouchScreen->getMode(&ts_mode);
if(err == ERR_SUCCESS)
{
    switch(ts_mode)
    {
        case MOUSE_NEXT_BOOT: cout << "USB profile is set to Mouse profile (active
            next boot)" << endl; break;
    }
}
```

```

    case TOUCH_NEXT_BOOT: cout << "USB profile is set to Touch profile (active
        next boot)" << endl; break;
    case MOUSE_NOW: cout << "USB profile is set to Mouse profile" << endl;
        break;
    case TOUCH_NOW: cout << "USB profile is set to Touch profile" << endl;
        break;
    default: cout << "Error: invalid setting returned from getMode" << endl;
        break;
    }
}
else
{
    cout << "Error(" << err << ") in function getMode: " << GetErrorStringA(err
        ) << endl;
}

```

5.20.2.3 virtual eErr CrossControl::TouchScreen::getMouseRightClickTime (unsigned short * *time*) [pure virtual]

Get mouse right click time. Applies only to the mouse profile. Use the OS settings for the touch profile.

Parameters

<i>time</i>	The right click time, in milliseconds.
-------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

err = pTouchScreen->getMouseRightClickTime(&rightclicktime);
if(err == ERR_SUCCESS)
{
    cout << "Right click time is set to: " << (int)rightclicktime << " ms" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getMouseRightClickTime: " <<
        GetErrorStringA(err) << endl;
}

```

5.20.2.4 virtual void CrossControl::TouchScreen::Release () [pure virtual]

Delete the [TouchScreen](#) object.

Returns

-

Example Usage:

```
TOUCHSCREENHANDLE pTouchScreen = ::GetTouchScreen();
assert (pTouchScreen);

touchscreen_example (pTouchScreen);

pTouchScreen->Release ();
```

5.20.2.5 virtual eErr CrossControl::TouchScreen::setAdvancedSetting (TSAdvancedSettingsParameter *param*, unsigned short *data*) [pure virtual]

Set advanced touch screen settings. See the description of TSAdvancedSettingsParameter for a description of the parameters.

Parameters

<i>param</i>	The setting to set.
<i>data</i>	The data value to set.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.20.2.6 virtual eErr CrossControl::TouchScreen::setMode (TouchScreenModeSettings *config*) [pure virtual]

Set Touch Screen mode. Sets the mode of the USB profile.

Parameters

<i>config</i>	The mode to set.
---------------	------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.20.2.7 virtual eErr CrossControl::TouchScreen::setMouseRightClickTime (unsigned short *time*) [pure virtual]

Set mouse right click time. Applies only to the mouse profile. Use the OS settings for the touch profile.

Parameters

<i>time</i>	The right click time, in milliseconds.
-------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/TouchScreen.h](#)

5.21 CrossControl::TouchScreenCalib Struct Reference

```
#include <TouchScreenCalib.h>
```

Public Member Functions

- virtual [eErr SetMode](#) ([CalibrationModeSettings](#) mode)=0
- virtual [eErr GetMode](#) ([CalibrationModeSettings](#) *mode)=0
- virtual [eErr SetCalibrationPoint](#) (unsigned char pointNr)=0
- virtual [eErr CheckCalibrationPointFinished](#) (bool *finished, unsigned char pointNr)=0
- virtual [eErr GetConfigParam](#) ([CalibrationConfigParam](#) param, unsigned short *value)=0
- virtual [eErr SetConfigParam](#) ([CalibrationConfigParam](#) param, unsigned short value)=0
- virtual void [Release](#) ()=0

5.21.1 Detailed Description

Touch Screen Calibration interface

Use the globally defined function [GetTouchScreenCalib\(\)](#) to get a handle to the [TouchScreenCalib](#) struct. Use the method [TouchScreenCalib::Release\(\)](#) to return the handle.

5.21.2 Member Function Documentation

5.21.2.1 virtual [eErr CrossControl::TouchScreenCalib::CheckCalibrationPointFinished](#) (bool * *finished*, unsigned char *pointNr*) [pure virtual]

Check if a calibration point is finished

Parameters

<i>finished</i>	Is current point finished?
<i>pointNr</i>	Calibration point number (1 to total number of points)

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.21.2.2 `virtual eErr CrossControl::TouchScreenCalib::GetConfigParam (CalibrationConfigParam param, unsigned short * value) [pure virtual]`

Get calibration config parameters

Parameters

<i>param</i>	Config parameter
<i>value</i>	Parameter value

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.21.2.3 `virtual eErr CrossControl::TouchScreenCalib::GetMode (CalibrationModeSettings * mode) [pure virtual]`

Get mode of front controller.

Parameters

<i>mode</i>	Current calibration mode
-------------	--------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.21.2.4 `virtual void CrossControl::TouchScreenCalib::Release () [pure virtual]`

Delete the [TouchScreenCalib](#) object.

Returns

-

5.21.2.5 `virtual eErr CrossControl::TouchScreenCalib::SetCalibrationPoint (unsigned char pointNr) [pure virtual]`

Set calibration point

Parameters

<i>pointNr</i>	Calibartion point number (1 to total number of points)
----------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.21.2.6 virtual eErr CrossControl::TouchScreenCalib::SetConfigParam (CalibrationConfigParam *param*, unsigned short *value*) [pure virtual]

Set calibration config parameters

Parameters

<i>param</i>	Config parameter
<i>value</i>	parameter value

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.21.2.7 virtual eErr CrossControl::TouchScreenCalib::SetMode (CalibrationModeSettings *mode*) [pure virtual]

Set mode of front controller.

Parameters

<i>mode</i>	Selected calibration mode
-------------	---------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/TouchScreenCalib.h](#)

5.22 CrossControl::UpgradeStatus Struct Reference

```
#include <CCAuxTypes.h>
```

Public Attributes

- enum [UpgradeAction](#) *currentAction*

- unsigned char [percent](#)
- [eErr](#) [errorCode](#)

5.22.1 Detailed Description

Upgrade Status

5.22.2 Member Data Documentation

5.22.2.1 [enum UpgradeAction](#) [CrossControl::UpgradeStatus::currentAction](#)

5.22.2.2 [eErr](#) [CrossControl::UpgradeStatus::errorCode](#)

Represents the percentage of completion of the current action

5.22.2.3 [unsigned char](#) [CrossControl::UpgradeStatus::percent](#)

The current action.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/CCAuxTypes.h](#)

5.23 CrossControl::version_info Struct Reference

```
#include <CCAuxTypes.h>
```

Public Attributes

- unsigned char [major](#)
- unsigned char [minor](#)
- unsigned char [release](#)
- unsigned char [build](#)

5.23.1 Member Data Documentation

5.23.1.1 [unsigned char](#) [CrossControl::version_info::build](#)

version build number

5.23.1.2 [unsigned char](#) [CrossControl::version_info::major](#)

version major number

5.23.1.3 unsigned char CrossControl::version_info::minor

version minor number

5.23.1.4 unsigned char CrossControl::version_info::release

version release number

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/CCAuxTypes.h](#)

5.24 CrossControl::Video Struct Reference

```
#include <Video.h>
```

Public Member Functions

- virtual [eErr init](#) (unsigned char deviceNr)=0
- virtual [eErr showVideo](#) (bool show)=0
- virtual [eErr setDeInterlaceMode](#) ([DeInterlaceMode](#) mode)=0
- virtual [eErr getDeInterlaceMode](#) ([DeInterlaceMode](#) *mode)=0
- virtual [eErr setMirroring](#) ([CCStatus](#) mode)=0
- virtual [eErr getMirroring](#) ([CCStatus](#) *mode)=0
- virtual [eErr setActiveChannel](#) ([VideoChannel](#) channel)=0
- virtual [eErr getActiveChannel](#) ([VideoChannel](#) *channel)=0
- virtual [eErr setColorKeys](#) (unsigned char rKey, unsigned char gKey, unsigned char bKey)=0
- virtual [eErr getColorKeys](#) (unsigned char *rKey, unsigned char *gKey, unsigned char *bKey)=0
- virtual [eErr setVideoArea](#) (unsigned short topLeftX, unsigned short topLeftY, unsigned short bottomRigthX, unsigned short bottomRigthY)=0
- virtual [eErr getVideoArea](#) (unsigned short *topLeftX, unsigned short *topLeftY, unsigned short *bottomRightX, unsigned short *bottomRightY)=0
- virtual [eErr getRawImage](#) (unsigned short *width, unsigned short *height, float *frameRate)=0
- virtual [eErr getVideoStandard](#) ([videoStandard](#) *standard)=0
- virtual [eErr getStatus](#) (unsigned char *status)=0
- virtual [eErr setScaling](#) (float x, float y)=0
- virtual [eErr getScaling](#) (float *x, float *y)=0
- virtual [eErr activateSnapshot](#) (bool activate)=0
- virtual [eErr takeSnapshot](#) (const char *path, bool bInterlaced)=0
- virtual [eErr takeSnapshotRaw](#) (char *rawImgBuffer, unsigned long rawImgBuff-Size, bool bInterlaced)=0
- virtual [eErr takeSnapshotBmp](#) (char **bmpBuffer, unsigned long *bmpBufSize, bool bInterlaced, bool bNTSCFormat)=0

- virtual [eErr createBitmap](#) (char **bmpBuffer, unsigned long *bmpBufSize, const char *rawImgBuffer, unsigned long rawImgBufSize, bool bInterlaced, bool bNTSCFormat)=0
- virtual [eErr freeBmpBuffer](#) (char *bmpBuffer)=0
- virtual [eErr minimize](#) ()=0
- virtual [eErr restore](#) ()=0
- virtual [eErr setDecoderReg](#) (unsigned char decoderRegister, unsigned char registerValue)=0
- virtual [eErr getDecoderReg](#) (unsigned char decoderRegister, unsigned char *registerValue)=0
- virtual [eErr setCropping](#) (unsigned char top, unsigned char left, unsigned char bottom, unsigned char right)=0
- virtual [eErr getCropping](#) (unsigned char *top, unsigned char *left, unsigned char *bottom, unsigned char *right)=0
- virtual void [Release](#) ()=0

5.24.1 Detailed Description

Analog [Video](#)

Use the globally defined function [GetVideo\(\)](#) to get a handle to the [Video](#) struct. Use the method [Video::Release\(\)](#) to return the handle.

5.24.2 Member Function Documentation

5.24.2.1 virtual eErr CrossControl::Video::activateSnapshot (bool *activate*)
[pure virtual]

To be able to take snapshot the snapshot function has to be active. After activation it takes 120ms before first snapshot can be taken. The Snapshot function can be active all the time. If power consumption and heat is an issue, snapshot may be turned off.

Parameters

<i>activate</i>	Set to true if the snapshot function shall be active.
-----------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.2 virtual eErr CrossControl::Video::createBitmap (char ** *bmpBuffer*, unsigned long * *bmpBufSize*, const char * *rawImgBuffer*, unsigned long *rawImgBufSize*, bool *bInterlaced*, bool *bNTSCFormat*) [pure virtual]

Create a bitmap from a raw image buffer. The bmp buffer is allocated in the function and has to be deallocated by the application.

Parameters

<i>bmpBuffer</i>	Bitmap ram buffer allocated by the API, has to be deallocated with freeBmpBuffer() by the application.
<i>bmpBufSize</i>	Size of the returned bitmap buffer.
<i>rawImg-Buffer</i>	Raw image buffer from takeSnapShotRaw.
<i>rawImgBuf-Size</i>	Size of the raw image buffer.
<i>bInterlaced</i>	Interlaced, if true the bitmap only contains every second line in the image, to save bandwidth.
<i>bNTSC-Format</i>	True if the video format in rawImageBuffer is NTSC format.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.3 virtual eErr CrossControl::Video::freeBmpBuffer (char * *bmpBuffer*) [pure virtual]

Free the memory allocated for BMP buffer.

Parameters

<i>bmpBuffer</i>	The bmp buffer to free.
------------------	-------------------------

Returns

error status.

5.24.2.4 virtual eErr CrossControl::Video::getActiveChannel (VideoChannel * *channel*) [pure virtual]

Get the current video channel.

Parameters

<i>channel</i>	Enum defining available channels.
----------------	-----------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.5 `virtual eErr CrossControl::Video::getColorKeys (unsigned char * rKey, unsigned char * gKey, unsigned char * bKey) [pure virtual]`

Get color key values. Note that the system uses 18 bit colors, so the two least significant bits are not used.

Parameters

<i>rKey</i>	Red value.
<i>gKey</i>	Green value.
<i>bKey</i>	Blue value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.6 `virtual eErr CrossControl::Video::getCropping (unsigned char * top, unsigned char * left, unsigned char * bottom, unsigned char * right) [pure virtual]`

Get Crop parameters.

Parameters

<i>top</i>	Crop top (lines).
<i>left</i>	Crop left (lines).
<i>bottom</i>	Crop bottom (lines).
<i>right</i>	Crop right (lines).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.7 `virtual eErr CrossControl::Video::getDecoderReg (unsigned char decoderRegister, unsigned char * registerValue) [pure virtual]`

Get the [Video](#) decoder bus register. Advanced function for direct access to the video decoder TVP5150AM1 registers.

Parameters

<i>decoder-Register</i>	Decoder Register Address.
-------------------------	---------------------------

<i>register-Value</i>	register value.
-----------------------	-----------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.8 virtual eErr CrossControl::Video::getDeInterlaceMode (DeInterlaceMode * *mode*) [pure virtual]

Get the deinterlace mode used when decoding the interlaced video stream.

Parameters

<i>mode</i>	The current mode. See enum DeInterlaceMode for descriptions of the modes.
-------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.9 virtual eErr CrossControl::Video::getMirroring (CCStatus * *mode*) [pure virtual]

Get the current mirroring mode of the video image.

Parameters

<i>mode</i>	The current mode. Enabled or Disabled.
-------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.10 virtual eErr CrossControl::Video::getRawImage (unsigned short * *width*, unsigned short * *height*, float * *frameRate*) [pure virtual]

Get the raw image size of moving image before any scaling and frame rate. For snapshot the height is 4 row less.

Parameters

<i>width</i>	Width of raw image.
<i>height</i>	Height of raw moving image, snapshot are 4 bytes less.
<i>frameRate</i>	Received video frame rate.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.11 `virtual eErr CrossControl::Video::getScaling (float * x, float * y)` [pure virtual]

Get [Video](#) Scaling (image size). If the deinterlace mode is set to DeInterlace_Even or DeInterlace_Odd, this function divides the actual vertical scaling by a factor of two, to get the same scaling factor as set with setScaling.

Parameters

<i>x</i>	Horizontal scaling (0.25-4).
<i>y</i>	Vertical scaling (0.25-4 DeInterlace_BOB) (0.125-2 DeInterlace_Even, DeInterlace_Odd).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.12 `virtual eErr CrossControl::Video::getStatus (unsigned char * status)` [pure virtual]

[Video](#) status byte.

Parameters

<i>status</i>	Status byte Bit 0: video on/off 0 = Off, 1 = On. Bit 2-1: De-interlacing method, 0 = Only even rows, 1 = Only odd rows, 2 = BOB, 3 = invalid. Bit 3: Mirroring mode, 0 = Off, 1 = On Bit 4: Read or write operation to analogue video decoder in progress. Bit 5: Analogue video decoder ready bit.
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.13 `virtual eErr CrossControl::Video::getVideoArea (unsigned short * topLeftX, unsigned short * topLeftY, unsigned short * bottomRightX, unsigned short * bottomRightY)` [pure virtual]

Get the area where video is shown.

Parameters

<i>topLeftX</i>	Top left X coordinate on screen.
<i>topLeftY</i>	Top left Y coordinate on screen.
<i>bottom-RightX</i>	Bottom right X coordinate on screen.
<i>bottom-RightY</i>	Bottom right Y coordinate on screen.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.14 `virtual eErr CrossControl::Video::getVideoStandard (videoStandard * standard) [pure virtual]`

Get video standard. The video decoder auto detects the video standard of the source.

Parameters

<i>standard</i>	Video standard.
-----------------	-----------------

Returns

error status.

5.24.2.15 `virtual eErr CrossControl::Video::init (unsigned char deviceNr) [pure virtual]`

Initialize a video device. The video device will initially use the following settings: DeInterlace_BOB and mirroring disabled.

Parameters

<i>deviceNr</i>	Device to connect to (1,2). Select one of 2 devices to connect to.
-----------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.16 `virtual eErr CrossControl::Video::minimize () [pure virtual]`

Minimizes the video area. Restore with `restore()` call.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.17 `virtual void CrossControl::Video::Release ()` [pure virtual]

Delete the [Video](#) object.

Returns

-

5.24.2.18 `virtual eErr CrossControl::Video::restore ()` [pure virtual]

Restores the video area to the size it was before a [minimize\(\)](#) call. Don't use restore if minimize has not been used first.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.19 `virtual eErr CrossControl::Video::setActiveChannel (VideoChannel channel)` [pure virtual]

Sets the active video channel.

Parameters

<i>channel</i>	Enum defining available channels.
----------------	-----------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.20 `virtual eErr CrossControl::Video::setColorKeys (unsigned char rKey, unsigned char gKey, unsigned char bKey)` [pure virtual]

Set color keys. Writes RGB color key values. Note that the system uses 18 bit colors, so the two least significant bits are not used.

Parameters

<i>rKey</i>	Red key value.
<i>gKey</i>	Green key value.
<i>bKey</i>	Blue key value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.21 `virtual eErr CrossControl::Video::setCropping (unsigned char top, unsigned char left, unsigned char bottom, unsigned char right)` [pure virtual]

Crop video image. Note that the video chip manual says the following about horizontal cropping: The number of pixels of active video must be an even number. The parameters *top* and *bottom* are internally converted to an even number. This is due to the input video being interlaced, a pair of odd/even lines are always cropped together.

Parameters

<i>top</i>	Crop top (0-255 lines).
<i>left</i>	Crop left (0-127 lines).
<i>bottom</i>	Crop bottom (0-255 lines).
<i>right</i>	Crop right (0-127 lines).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.22 `virtual eErr CrossControl::Video::setDecoderReg (unsigned char decoderRegister, unsigned char registerValue)` [pure virtual]

Set the [Video](#) decoder bus register. Advanced function for direct access to the video decoder TVP5150AM1 registers.

Parameters

<i>decoder-Register</i>	Decoder Register Address.
<i>register-Value</i>	register value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.23 `virtual eErr CrossControl::Video::setDeInterlaceMode (DeInterlaceMode mode)` [pure virtual]

Set the deinterlace mode used when decoding the interlaced video stream.

Parameters

<i>mode</i>	The mode to set. See enum DeInterlaceMode for descriptions of the modes.
-------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.24 `virtual eErr CrossControl::Video::setMirroring (CCStatus mode)`
`[pure virtual]`

Enable or disable mirroring of the video image.

Parameters

<i>mode</i>	The mode to set. Enabled or Disabled.
-------------	---------------------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.25 `virtual eErr CrossControl::Video::setScaling (float x, float y)` `[pure virtual]`

Set [Video](#) Scaling (image size). If the deinterlace mode is set to DeInterlace_Even or DeInterlace_Odd, this function multiplies the vertical scaling by a factor of two, to get the correct image proportions.

Parameters

<i>x</i>	Horizontal scaling (0.25-4).
<i>y</i>	Vertical scaling (0.25-4 DeInterlace_BOB) (0.125-2 DeInterlace_Even, DeInterlace_Odd).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.26 `virtual eErr CrossControl::Video::setVideoArea (unsigned short topLeftX, unsigned short topLeftY, unsigned short bottomRigthX, unsigned short bottomRigthY)` `[pure virtual]`

Set the area where video is shown.

Parameters

<i>topLeftX</i>	Top left X coordinate on screen.
<i>topLeftY</i>	Top left Y coordinate on screen.
<i>bottom-RigthX</i>	Bottom right X coordinate on screen.
<i>bottom-RigthY</i>	Bottom right Y coordinate on screen.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.27 `virtual eErr CrossControl::Video::showVideo (bool show) [pure virtual]`

Show or hide the video image. Note that it may take some time before the video is shown and correct input info can be read by `getRawImage`.

Parameters

<i>show</i>	True shows the video image.
-------------	-----------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.28 `virtual eErr CrossControl::Video::takeSnapshot (const char * path, bool bInterlaced) [pure virtual]`

Takes a snapshot of the current video image and stores it to a bitmap file. This is a combination of `takeSnapShotRaw`, `getVideoStandard` and `createBitMap` and then storing of the `bmpBuffer` to file. To be able to take a snapshot, the snapshot function has to be active.

Parameters

<i>path</i>	The file path to where the image should be stored.
<i>bInterlaced</i>	If true the bitmap only contains every second line in the image, to save bandwidth.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.29 `virtual eErr CrossControl::Video::takeSnapshotBmp (char ** bmpBuffer, unsigned long * bmpBufSize, bool bInterlaced, bool bNTSCFormat) [pure virtual]`

Takes a snapshot of the current video image and return a data buffer with a bitmap image. The bmp buffer is allocated in the function and has to be deallocated with `freeBmpBuffer()` by the application. This is a combination of the function `takeSnapShotRaw` and `createBitMap`. To be able to take a snapshot, the snapshot function has to be active.

Parameters

<i>bmpBuffer</i>	Bitmap ram buffer allocated by the API, has to be deallocated with freeBmpBuffer() by the application.
<i>bmpBufSize</i>	Size of the returned bitmap buffer.
<i>bInterlaced</i>	If true the bitmap only contains every second line in the image, to save bandwidth.
<i>bNTSC-Format</i>	True if the video format in rawImageBuffer is NTSC format.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.24.2.30 virtual eErr CrossControl::Video::takeSnapshotRaw (char * rawImgBuffer, unsigned long rawImgBuffSize, bool bInterlaced) [pure virtual]

Takes a snapshot of the current video image and return raw image data. The size of the raw image is when interlaced = false $0x100 + \text{line count} * \text{row count} * 4$. The size of the raw image is when interlaced = true $0x100 + \text{line count} * \text{row count} * 2$. To be able to take a snapshot, the snapshot function has to be active. This function is blocking until a new frame is available from the decoder. An error will be returned if the decoder doesn't return any frames before a timeout.

Parameters

<i>rawImg-Buffer</i>	Buffer for image to be stored in.
<i>rawImgBuff-Size</i>	Size of the buffer.
<i>bInterlaced</i>	If true the bitmap only contains every second line in the image, to save bandwidth.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

The documentation for this struct was generated from the following file:

- [fixedIncludeFiles/Video.h](#)

5.25 CrossControl::video_dec_command Struct Reference

```
#include <CCAuxTypes.h>
```


Public Attributes

- unsigned char [decoder_register](#)
- unsigned char [register_value](#)

5.25.1 Member Data Documentation

5.25.1.1 unsigned char `CrossControl::video_dec_command::decoder_register`

5.25.1.2 unsigned char `CrossControl::video_dec_command::register_value`

The documentation for this struct was generated from the following file:

- `fixedIncludeFiles/CCAuxTypes.h`

Chapter 6

File Documentation

6.1 fixedIncludeFiles/About.h File Reference

Classes

- struct [CrossControl::About](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::About](#) * [ABOUTHANDLE](#)

Functions

- `EXTERN_C CCAUXDLL_API ABOUTHANDLE CCAUXDLL_CALLING_CONV GetAbout (void)`

6.1.1 Typedef Documentation

6.1.1.1 typedef [CrossControl::About](#)* [ABOUTHANDLE](#)

6.1.2 Function Documentation

6.1.2.1 `EXTERN_C CCAUXDLL_API ABOUTHANDLE CCAUXDLL_CALLING_CONV GetAbout (void)`

Factory function that creates instances of the About object.

Returns

ABOUTHANDLE to an allocated About structure. The returned handle needs to be deallocated using the About::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
ABOUTHANDLE pAbout = ::GetAbout();
assert(pAbout);

list_about_information(pAbout);

pAbout->Release();
```

6.2 fixedIncludeFiles/Adc.h File Reference

Classes

- struct [CrossControl::Adc](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Adc](#) * ADCHANDLE

Enumerations

- enum [CrossControl::VoltageEnum](#) { [CrossControl::VOLTAGE_24VIN](#) = 0, [CrossControl::VOLTAGE_24V](#), [CrossControl::VOLTAGE_12V](#), [CrossControl::VOLTAGE_12VID](#), [CrossControl::VOLTAGE_5V](#), [CrossControl::VOLTAGE_3V3](#), [CrossControl::VOLTAGE_VTFT](#), [CrossControl::VOLTAGE_5VSTB](#), [CrossControl::VOLTAGE_1V9](#), [CrossControl::VOLTAGE_1V8](#), [CrossControl::VOLTAGE_1V5](#), [CrossControl::VOLTAGE_1V2](#), [CrossControl::VOLTAGE_1V05](#), [CrossControl::VOLTAGE_1V0](#), [CrossControl::VOLTAGE_0V9](#), [CrossControl::VOLTAGE_VREF_INT](#) }

Functions

- EXTERN_C CCAUXDLL_API ADCHANDLE CCAUXDLL_CALLING_CONV [GetAdc](#) (void)

6.2.1 Typedef Documentation

6.2.1.1 typedef CrossControl::Adc* ADCHANDLE

6.2.2 Function Documentation

6.2.2.1 EXTERN_C CCAUXDLL_API ADCHANDLE CCAUXDLL_CALLING_CONV GetAdc (void)

Factory function that creates instances of the Adc object.

Returns

ABOUTHANDLE to an allocated Adc structure. The returned handle needs to be deallocated using the Adc::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
ADCHANDLE pAdc = ::GetAdc();
assert (pAdc);

output_voltage (pAdc, "24VIN", CrossControl::VOLTAGE_24VIN);
output_voltage (pAdc, "24V", CrossControl::VOLTAGE_24V);
output_voltage (pAdc, "12V", CrossControl::VOLTAGE_12V);
output_voltage (pAdc, "12VID", CrossControl::VOLTAGE_12VID);
output_voltage (pAdc, "5V", CrossControl::VOLTAGE_5V);
output_voltage (pAdc, "3V3", CrossControl::VOLTAGE_3V3);
output_voltage (pAdc, "VTFT", CrossControl::VOLTAGE_VTFT);
output_voltage (pAdc, "5VSTB", CrossControl::VOLTAGE_5VSTB);
output_voltage (pAdc, "1V9", CrossControl::VOLTAGE_1V9);
output_voltage (pAdc, "1V8", CrossControl::VOLTAGE_1V8);
output_voltage (pAdc, "1V5", CrossControl::VOLTAGE_1V5);
output_voltage (pAdc, "1V2", CrossControl::VOLTAGE_1V2);
output_voltage (pAdc, "1V05", CrossControl::VOLTAGE_1V05);
output_voltage (pAdc, "1V0", CrossControl::VOLTAGE_1V0);
output_voltage (pAdc, "0V9", CrossControl::VOLTAGE_0V9);

pAdc->Release();
```

6.3 fixedIncludeFiles/AuxVersion.h File Reference

Classes

- struct [CrossControl::AuxVersion](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::AuxVersion](#) * [AUXVERSIONHANDLE](#)

Functions

- [EXTERN_C](#) [CCAUXDLL_API](#) [AUXVERSIONHANDLE](#) [CCAUXDLL_CALLING_CONV](#) [GetAuxVersion](#) (void)

6.3.1 Typedef Documentation

6.3.1.1 typedef [CrossControl::AuxVersion](#)* [AUXVERSIONHANDLE](#)

6.3.2 Function Documentation

6.3.2.1 [EXTERN_C](#) [CCAUXDLL_API](#) [AUXVERSIONHANDLE](#) [CCAUXDLL_CALLING_CONV](#) [GetAuxVersion](#) (void)

Factory function that creates instances of the [AuxVersion](#) object.

Returns

[AUXVERSIONHANDLE](#) to an allocated [AuxVersion](#) structure. The returned handle needs to be deallocated using the [AuxVersion::Release\(\)](#) method when it's no longer needed. Returns [NULL](#) if it fails to allocate memory.

Example Usage:

```
AUXVERSIONHANDLE pAuxVersion = ::GetAuxVersion();
assert (pAuxVersion);

output_versions (pAuxVersion);

pAuxVersion->Release();
```

6.4 fixedIncludeFiles/Backlight.h File Reference

Classes

- struct [CrossControl::Backlight](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Backlight](#) * [BACKLIGHTHANDLE](#)

Functions

- [EXTERN_C CCAUXDLL_API BACKLIGHTHANDLE CCAUXDLL_CALLING_CONV GetBacklight](#) (void)

6.4.1 Typedef Documentation

6.4.1.1 typedef [CrossControl::Backlight](#)* [BACKLIGHTHANDLE](#)

6.4.2 Function Documentation

6.4.2.1 [EXTERN_C CCAUXDLL_API BACKLIGHTHANDLE CCAUXDLL_CALLING_CONV GetBacklight](#) (void)

Factory function that creates instances of the Backlight object.

Returns

[BACKLIGHTHANDLE](#) to an allocated Backlight structure. The returned handle needs to be deallocated using the [Backlight::Release\(\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
BACKLIGHTHANDLE pBacklight = ::GetBacklight();
assert (pBacklight);

change_backlight (pBacklight);

pBacklight->Release();
```

6.5 fixedIncludeFiles/Buzzer.h File Reference

Classes

- struct [CrossControl::Buzzer](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Buzzer](#) * BUZZERHANDLE

Functions

- EXTERN_C CCAUXDLL_API [BUZZERHANDLE](#) CCAUXDLL_CALLING_CONV [GetBuzzer](#) (void)

6.5.1 Typedef Documentation

6.5.1.1 typedef [CrossControl::Buzzer](#)* BUZZERHANDLE

6.5.2 Function Documentation

6.5.2.1 EXTERN_C CCAUXDLL_API [BUZZERHANDLE](#) CCAUXDLL_CALLING_CONV [GetBuzzer](#) (void)

Factory function that creates instances of the Buzzer object.

Returns

BUZZERHANDLE to an allocated Buzzer structure. The returned handle needs to be deallocated using the [Buzzer::Release\(\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
BUZZERHANDLE pBuzzer = ::GetBuzzer();
assert (pBuzzer);

play_beeps (pBuzzer);

pBuzzer->Release();
```

6.6 fixedIncludeFiles/CanSetting.h File Reference

Classes

- struct [CrossControl::CanSetting](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::CanSetting](#) * [CANSETTINGHANDLE](#)

Functions

- EXTERN_C CCAUXDLL_API [CANSETTINGHANDLE](#) CCAUXDLL_CALLING_CONV [GetCanSetting](#) (void)

6.6.1 Typedef Documentation

6.6.1.1 typedef [CrossControl::CanSetting](#)* [CANSETTINGHANDLE](#)

6.6.2 Function Documentation

6.6.2.1 EXTERN_C CCAUXDLL_API [CANSETTINGHANDLE](#) CCAUXDLL_CALLING_CONV [GetCanSetting](#) (void)

Factory function that creates instances of the [CanSetting](#) object.

Returns

[CANSETTINGHANDLE](#) to an allocated [CanSetting](#) structure. The returned handle needs to be deallocated using the [CanSetting::Release\(\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
CANSETTINGHANDLE pCanSetting = ::GetCanSetting();
assert (pCanSetting);

read_cansettings (pCanSetting);

pCanSetting->Release();
```

6.7 fixedIncludeFiles/CCAuxErrors.h File Reference

Namespaces

- namespace [CrossControl](#)

Functions

- EXTERN_C CCAUXDLL_API char const *CCAUXDLL_CALLING_CONV [CrossControl::GetErrorStringA](#) (eErr errCode)
- EXTERN_C CCAUXDLL_API wchar_t const *CCAUXDLL_CALLING_CONV [CrossControl::GetErrorStringW](#) (eErr errCode)

6.8 fixedIncludeFiles/CCAuxTypes.h File Reference

Classes

- struct `CrossControl::received_video`
- struct `CrossControl::video_dec_command`
- struct `CrossControl::version_info`
- struct `CrossControl::BuzzerSetup`
- struct `CrossControl::LedTimingType`
- struct `CrossControl::LedColorMixType`
- struct `CrossControl::TimerType`
- struct `CrossControl::UpgradeStatus`

Namespaces

- namespace `CrossControl`

Typedefs

- typedef struct version_info `CrossControl::VersionType`

Enumerations

- enum `CrossControl::LightSensorOperationRange` { `CrossControl::RangeStandard` = 0, `CrossControl::RangeExtended` = 1 }
- enum `CrossControl::LightSensorSamplingMode` { `CrossControl::SamplingModeStandard` = 0, `CrossControl::SamplingModeExtended`, `CrossControl::SamplingModeAuto` }
- enum `CrossControl::CCStatus` { `CrossControl::Disabled` = 0, `CrossControl::Enabled` = 1 }
- enum `CrossControl::eErr` { `CrossControl::ERR_SUCCESS` = 0, `CrossControl::ERR_OPEN_FAILED` = 1, `CrossControl::ERR_NOT_SUPPORTED` = 2, `CrossControl::ERR_UNKNOWN_FEATURE` = 3, `CrossControl::ERR_DATATYPE_MISMATCH` = 4, `CrossControl::ERR_CODE_NOT_EXIST` = 5, `CrossControl::ERR_BUFFER_SIZE` = 6, `CrossControl::ERR_IOCTL_FAILED` = 7, `CrossControl::ERR_INVALID_DATA` = 8, `CrossControl::ERR_INVALID_PARAMETER` = 9, `CrossControl::ERR_CREATE_THREAD` = 10, `CrossControl::ERR_IN_PROGRESS` = 11, `CrossControl::ERR_CHECKSUM` = 12, `CrossControl::ERR_INIT_FAILED` = 13, `CrossControl::ERR_VERIFY_FAILED` = 14, `CrossControl::ERR_DEVICE_READ_DATA_FAILED` = 15, `CrossControl::ERR_DEVICE_WRITE_DATA_FAILED` = 16, `CrossControl::ERR_COMMAND_FAILED` = 17, `CrossControl::ERR_EEPROM` = 18, `CrossControl::ERR_JIDA_TEMP` = 19, `CrossControl::ERR_AVERAGE_CALC_STARTED` = 20,

- CrossControl::ERR_NOT_RUNNING = 21, CrossControl::ERR_I2C_EXPANDER_READ_FAILED = 22, CrossControl::ERR_I2C_EXPANDER_WRITE_FAILED = 23, CrossControl::ERR_I2C_EXPANDER_INIT_FAILED = 24, CrossControl::ERR_NEWER_SS_VERSION_REQUIRED = 25, CrossControl::ERR_NEWER_FPGA_VERSION_REQUIRED = 26, CrossControl::ERR_NEWER_FRONT_VERSION_REQUIRED = 27, CrossControl::ERR_TELEMATICS_GPRS_NOT_AVAILABLE = 28, CrossControl::ERR_TELEMATICS_WLAN_NOT_AVAILABLE = 29, CrossControl::ERR_TELEMATICS_BT_NOT_AVAILABLE = 30, CrossControl::ERR_TELEMATICS_GPS_NOT_AVAILABLE = 31, CrossControl::ERR_MEM_ALLOC_FAIL = 32 }
- enum CrossControl::DeInterlaceMode { CrossControl::DeInterlace_Even = 0, CrossControl::DeInterlace_Odd = 1, CrossControl::DeInterlace_BOB = 2 }
 - enum CrossControl::VideoChannel { CrossControl::Analog_Channel_1 = 0, CrossControl::Analog_Channel_2 = 1, CrossControl::Analog_Channel_3 = 2, CrossControl::Analog_Channel_4 = 3 }
 - enum CrossControl::videoStandard { CrossControl::STD_M_J_NTSC = 0, CrossControl::STD_B_D_G_H_I_N_PAL = 1, CrossControl::STD_M_PAL = 2, CrossControl::STD_PAL = 3, CrossControl::STD_NTSC = 4, CrossControl::STD_SECAM = 5 }
 - enum CrossControl::CanFrameType { CrossControl::FrameStandard, CrossControl::FrameExtended, CrossControl::FrameStandardExtended }
 - enum CrossControl::TriggerConf { CrossControl::Front_Button_Enabled = 1, CrossControl::OnOff_Signal_Enabled = 2, CrossControl::Both_Button_And_Signal_Enabled = 3 }
 - enum CrossControl::PowerAction { CrossControl::NoAction = 0, CrossControl::ActionSuspend = 1, CrossControl::ActionShutDown = 2 }
 - enum CrossControl::ButtonPowerTransitionStatus { CrossControl::BPTS_No_Change = 0, CrossControl::BPTS_ShutDown = 1, CrossControl::BPTS_Suspend = 2, CrossControl::BPTS_Restart = 3, CrossControl::BPTS_BtnPressed = 4, CrossControl::BPTS_BtnPressedLong = 5, CrossControl::BPTS_SignalOff = 6 }
 - enum CrossControl::JidaSensorType { CrossControl::TEMP_CPU = 0, CrossControl::TEMP_BOX = 1, CrossControl::TEMP_ENV = 2, CrossControl::TEMP_BOARD = 3, CrossControl::TEMP_BACKPLANE = 4, CrossControl::TEMP_CHIPSETS = 5, CrossControl::TEMP_VIDEO = 6, CrossControl::TEMP_OTHER = 7 }
 - enum CrossControl::UpgradeAction { CrossControl::UPGRADE_INIT, CrossControl::UPGRADE_PREP_COM, CrossControl::UPGRADE_READING_FILE, CrossControl::UPGRADE_CONVERTING_FILE, CrossControl::UPGRADE_FLASHING, CrossControl::UPGRADE_VERIFYING, CrossControl::UPGRADE_COMPLETE, CrossControl::UPGRADE_COMPLETE_WITH_ERRORS }
 - enum CrossControl::CCAuxColor { CrossControl::RED = 0, CrossControl::GREEN, CrossControl::BLUE, CrossControl::CYAN, CrossControl::MAGENTA, CrossControl::YELLOW, CrossControl::UNDEFINED_COLOR }

6.9 fixedIncludeFiles/CCPlatform.h File Reference

6.10 fixedIncludeFiles/Config.h File Reference

Classes

- struct [CrossControl::Config](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Config](#) * [CONFIGHANDLE](#)

Functions

- `EXTERN_C CCAUXDLL_API CONFIGHANDLE CCAUXDLL_CALLING_CONV GetConfig (void)`

Variables

- const unsigned char [CrossControl::Video1Conf](#) = (1 << 0)
- const unsigned char [CrossControl::Video2Conf](#) = (1 << 1)
- const unsigned char [CrossControl::Video3Conf](#) = (1 << 2)
- const unsigned char [CrossControl::Video4Conf](#) = (1 << 3)

6.10.1 Typedef Documentation

6.10.1.1 typedef [CrossControl::Config](#)* [CONFIGHANDLE](#)

6.10.2 Function Documentation

6.10.2.1 `EXTERN_C CCAUXDLL_API CONFIGHANDLE CCAUXDLL_CALLING_CONV GetConfig (void)`

Factory function that creates instances of the Config object.

Returns

[CONFIGHANDLE](#) to an allocated Config structure. The returned handle needs to be deallocated using the [Config::Release\(\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
CONFIGHANDLE pConfig = ::GetConfig();
assert(pConfig);

conf_example(pConfig);

pConfig->Release();
```

6.11 fixedIncludeFiles/Diagnostic.h File Reference

Classes

- struct [CrossControl::Diagnostic](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Diagnostic](#) * [DIAGNOSTICHANDLE](#)

Functions

- [EXTERN_C CCAUXDLL_API DIAGNOSTICHANDLE CCAUXDLL_CALLING_CONV GetDiagnostic](#) (void)

6.11.1 Typedef Documentation

6.11.1.1 typedef [CrossControl::Diagnostic](#)* [DIAGNOSTICHANDLE](#)

6.11.2 Function Documentation

6.11.2.1 [EXTERN_C CCAUXDLL_API DIAGNOSTICHANDLE CCAUXDLL_CALLING_CONV GetDiagnostic](#) (void)

Factory function that creates instances of the Diagnostic object.

Returns

[DIAGNOSTICHANDLE](#) to an allocated [Diagnostic](#) structure. The returned handle needs to be deallocated using the [Diagnostic::Release\(\)](#) method when it's no longer needed. Returns [NULL](#) if it fails to allocate memory.

Example Usage:

```

DIAGNOSTICHANDLE pDiagnostic = ::GetDiagnostic();
assert(pDiagnostic);

diagnostic_example(pDiagnostic);

pDiagnostic->Release();

```

6.12 fixedIncludeFiles/DiagnosticCodes.h File Reference

Namespaces

- namespace [CrossControl](#)

Enumerations

- enum [CrossControl::startupReasonCodes](#) { [CrossControl::startupReasonCodeUndefined](#) = 0x0000, [CrossControl::startupReasonCodeButtonPress](#) = 0x0055, [CrossControl::startupReasonCodeExtCtrl](#) = 0x00AA, [CrossControl::startupReasonCodeMPRestart](#) = 0x00F0, [CrossControl::startupReasonCodePowerOnStartup](#) = 0x000F }
- enum [CrossControl::shutDownReasonCodes](#) { [CrossControl::shutdownReasonCodeNoError](#) = 0x001F }
- enum [CrossControl::hwErrorStatusCodes](#) { [CrossControl::errCodeNoErr](#) = 0, [CrossControl::errCodeFPGACONFReadErr](#) = 1, [CrossControl::errCodeFPGACONFUnexpVal](#) = 2, [CrossControl::errCodeCBRESETReadErr](#) = 3, [CrossControl::errCodeSUS3ReadErr](#) = 4, [CrossControl::errCodeSUS4ReadErr](#) = 5, [CrossControl::errCodeSUS5ReadErr](#) = 6, [CrossControl::errCodePG5VSTBYReadErr](#) = 7, [CrossControl::errCodePG5VSTBYUnexpVal](#) = 8, [CrossControl::errCodeCANPWROKReadErr](#) = 9, [CrossControl::errCodeVIDPWROKReadErr](#) = 10, [CrossControl::errCodeLVDSBLENReadErr](#) = 11, [CrossControl::errCodeLVDVDDENReadErr](#) = 12, [CrossControl::errCodeEXTCTRLONReadErr](#) = 13, [CrossControl::errCodeFPBTNONReadErr](#) = 14, [CrossControl::errCode24VReadErr](#) = 15, [CrossControl::errCode24VOutOfLimits](#) = 16, [CrossControl::errCode24VINReadErr](#) = 17, [CrossControl::errCode24VINOutOfLimits](#) = 18, [CrossControl::errCode12VReadErr](#) = 19, [CrossControl::errCode12VOutOfLimits](#) = 20, [CrossControl::errCode12VVIDEOReadErr](#) = 21, [CrossControl::errCode12VVIDEOOutOfLimits](#) = 22, [CrossControl::errCode5VSTBYReadErr](#) = 23, [CrossControl::errCode5VSTBYOutOfLimits](#) = 24, [CrossControl::errCode5VReadErr](#) = 25, [CrossControl::errCode5VOutOfLimits](#) = 26, [CrossControl::errCode3V3ReadErr](#) = 27, [CrossControl::errCode3V3OutOfLimits](#) = 28, [CrossControl::errCodeTFTVOLReadErr](#) = 29, [CrossControl::errCodeTFTVOLOOutOfLimits](#) = 30, [CrossControl::errCode1V9ReadErr](#) = 31, [CrossControl::errCode1V9OutOfLimits](#) = 32, [CrossControl::errCode1V8ReadErr](#) = 33, [CrossControl::errCode1V8OutOfLimits](#) = 34, [CrossControl::errCode1V5ReadErr](#) = 35, [CrossControl::errCode1V5OutOfLimits](#) = 36, [CrossControl::errCode1V2ReadErr](#) = 37, [CrossControl::errCode1V2OutOfLimits](#) = 38, [CrossControl::errCode1V05ReadErr](#) = 39, [CrossControl::errCode1V05OutOfLimits](#) = 40, [CrossControl::errCode1V0ReadErr](#) = 41, [CrossControl::errCode1V0OutOfLimits](#) = 42, [CrossControl::err-](#)

Code0V9ReadErr = 43, CrossControl::errCode0V9OutOfLimits = 44, CrossControl::errCodeI2CTEMPReadErr = 45, CrossControl::errCodeI2CTEMPOutOfLimits = 46, CrossControl::errCodeSTM32TEMPReadErr = 47, CrossControl::errCodeSTM32TEMPOutOfLimits = 48, CrossControl::errCodeBLTYPEUnexpEEPROMVal = 49, CrossControl::errCodeFPBTNUnexpEEPROMVal = 50, - CrossControl::errCodeEXTCTRLUnexpEEPROMVal = 51, CrossControl::errCodeLowRange24VUnexpEEPROMVal = 52, CrossControl::errCodeSuspToRAMUnexpEEPROMVal = 53, CrossControl::errCodeCANPWRUnexpEEPROMVal = 54, CrossControl::errCodeVID1PWRUnexpEEPROMVal = 55, CrossControl::errCodeVID2PWRUnexpEEPROMVal = 56, CrossControl::errCodeVID3PWRUnexpEEPROMVal = 57, CrossControl::errCodeVID4PWRUnexpEEPROMVal = 58, CrossControl::errCodeEXTFANUnexpEEPROMVal = 59, - CrossControl::errCodeLEDUnexpEEPROMVal = 60, CrossControl::errCodeUnitTypeUnexpEEPROMVal = 61, CrossControl::errCodeBLTYPEReadErrEEPROM = 62, CrossControl::errCodeFPBTNReadErrEEPROM = 63, CrossControl::errCodeEXTCTRLReadErrEEPROM = 64, CrossControl::errCodeMaxSuspTimeReadErrEEPROM = 65, CrossControl::errCodeLowRange24VReadErrEEPROM = 66, CrossControl::errCodeSuspToRAMReadErrEEPROM = 67, - CrossControl::errCodeCANPWRReadErrEEPROM = 68, CrossControl::errCodeVID1PWRReadErrEEPROM = 69, CrossControl::errCodeVID2PWRReadErrEEPROM = 70, CrossControl::errCodeVID3PWRReadErrEEPROM = 71, - CrossControl::errCodeVID4PWRReadErrEEPROM = 72, CrossControl::errCodeEXTFANReadErrEEPROM = 73, CrossControl::errCodeLEDReadErrEEPROM = 74, CrossControl::errCodeUnitTypeReadErrEEPROM = 75, CrossControl::errCodeRCCInit = 76, CrossControl::errCodeDriverInit = 77, CrossControl::errCodeSetSUPPLYRESET = 78, CrossControl::errCodeRelSUPPLYRESET = 79, CrossControl::errCodeSetSYSRESET = 80, CrossControl::errCodeRelSYSRESET = 81, CrossControl::errCodeSetPWRBTN = 82, CrossControl::errCodeRelPWRBTN = 83, CrossControl::errCodeOnBL = 84, CrossControl::errCodeOffBL = 85, CrossControl::errCodeEXTFANOn = 86, CrossControl::errCodeEXTFANOff = 87, CrossControl::errCodePWRENOn = 88, CrossControl::errCodePWRENOff = 89, CrossControl::errCodeMPPWRENOn = 90, CrossControl::errCodeMPPWRENOff = 91, CrossControl::errCodeCANPWRENOn = 92, CrossControl::errCodeCANPWRENOff = 93, CrossControl::errCodeVID1PWRENOn = 94, CrossControl::errCodeVID1PWRENOff = 95, × CrossControl::errCodeVID2PWRENOn = 96, CrossControl::errCodeVID2PWRENOff = 97, CrossControl::errCodeVID3PWRENOn = 98, CrossControl::errCodeVID3PWRENOff = 99, CrossControl::errCodeVID4PWRENOn = 100, - CrossControl::errCodeVID4PWRENOff = 101, CrossControl::errCodeHEATACTOn = 102, CrossControl::errCodeHEATACTOff = 103, CrossControl::errCodeSetLEDCol = 104, CrossControl::errCodeSetLEDFreq = 105, CrossControl::errCodeManageLED = 106, CrossControl::errCodeManageCANPwr = 107, - CrossControl::errCodeManageMPPwr = 108, CrossControl::errCodeManageVidPwr = 109, CrossControl::errCodeManagePowSup = 110, CrossControl::errCodeManageReset = 111, CrossControl::errCodeSSState = 112, CrossControl::errCodeVarWrapAround = 113, CrossControl::errCodeFPBTNUnexpVal = 114, - CrossControl::errCodeEXTCTRLUnexpVal = 115, CrossControl::errCodeMAINPWROKReadErr = 116, CrossControl::errCodeFRONTSPAREReadErr = 117, CrossControl::errCodeTIMERReadErr = 118, CrossControl::errCodeManageDiagnostics = 119, CrossControl::errCodeFPBTNTimOutReadErrEEPROM =

120, [CrossControl::errCodeEXTCTRLTimOutReadErrEEPROM](#) = 121, [CrossControl::errCodeFPBTNAndExtCtrlDisabled](#) = 122, [CrossControl::errCodeSWVerReadErr](#) = 123, [CrossControl::errCodeSWVerWriteErr](#) = 124, [CrossControl::errCodeManageActDeAct](#) = 125, [CrossControl::errCodeTickTimeOutTimer](#) = 126, [CrossControl::errCodeOperateModeStateError](#) = 127, [CrossControl::errCodeHeatingTempReadErrEEPROM](#) = 128, [CrossControl::errCodeMPFailedStart](#) = 129, [CrossControl::errCodeReadErrEEPROM](#) = 130, [CrossControl::errCodeTimeOutWaitingForVoltages](#) = 131, [CrossControl::errCodeMAX](#) }

Functions

- `EXTERN_C CCAUXDLL_API char const *CCAUXDLL_CALLING_CONV CrossControl::GetHwErrorStatusStringA (unsigned short errCode)`
- `EXTERN_C CCAUXDLL_API wchar_t const *CCAUXDLL_CALLING_CONV CrossControl::GetHwErrorStatusStringW (unsigned short errCode)`
- `EXTERN_C CCAUXDLL_API char const *CCAUXDLL_CALLING_CONV CrossControl::GetStartupReasonStringA (unsigned short code)`
- `EXTERN_C CCAUXDLL_API wchar_t const *CCAUXDLL_CALLING_CONV CrossControl::GetStartupReasonStringW (unsigned short code)`

6.13 fixedIncludeFiles/DigIO.h File Reference

Classes

- struct [CrossControl::DigIO](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::DigIO](#) * [DIGIOHANDLE](#)

Functions

- `EXTERN_C CCAUXDLL_API DIGIOHANDLE CCAUXDLL_CALLING_CONV GetDigIO (void)`

Variables

- const unsigned char [CrossControl::DigitalIn_1](#) = (1 << 0)
- const unsigned char [CrossControl::DigitalIn_2](#) = (1 << 1)
- const unsigned char [CrossControl::DigitalIn_3](#) = (1 << 2)
- const unsigned char [CrossControl::DigitalIn_4](#) = (1 << 3)

6.13.1 Typedef Documentation

6.13.1.1 typedef `CrossControl::DigIO*` DIGIOHANDLE

6.13.2 Function Documentation

6.13.2.1 EXTERN_C CCAUXDLL_API DIGIOHANDLE CCAUXDLL_CALLING_CONV `GetDigIO (void)`

Factory function that creates instances of the DigIO object.

Returns

DIGIOHANDLE to an allocated DigIO structure. The returned handle needs to be deallocated using the DigIO::Release() method when it's no longer needed. - Returns NULL if it fails to allocate memory.

Example Usage:

```
DIGIOHANDLE pDigIO = ::GetDigIO();  
assert (pDigIO);  
  
list_digital_inputs (pDigIO);  
  
pDigIO->Release();
```

6.14 fixedIncludeFiles/FirmwareUpgrade.h File Reference

Classes

- struct [CrossControl::FirmwareUpgrade](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::FirmwareUpgrade](#) * [FIRMWAREUPGHANDLE](#)

Functions

- EXTERN_C CCAUXDLL_API [FIRMWAREUPGHANDLE](#) CCAUXDLL_CALLING_CONV [GetFirmwareUpgrade](#) (void)

6.14.1 Typedef Documentation

6.14.1.1 `typedef CrossControl::FirmwareUpgrade* FIRMWAREUPGHANDLE`

6.14.2 Function Documentation

6.14.2.1 `EXTERN_C CCAUXDLL_API FIRMWAREUPGHANDLE CCAUXDLL_CALLING_CONV GetFirmwareUpgrade (void)`

Factory function that creates instances of the Adc object.

Returns

FIRMWAREUPGHANDLE to an allocated FirmwareUpgrade structure. The returned handle needs to be deallocated using the FirmwareUpgrade::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
FIRMWAREUPGHANDLE upgrade=GetFirmwareUpgrade();
assert (upgrade != NULL);
```

6.15 fixedIncludeFiles/FrontLED.h File Reference

Classes

- struct [CrossControl::FrontLED](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- `typedef CrossControl::FrontLED * FRONTLEDHANDLE`

Functions

- `EXTERN_C CCAUXDLL_API FRONTLEDHANDLE CCAUXDLL_CALLING_CONV GetFrontLED (void)`

6.15.1 Typedef Documentation

6.15.1.1 `typedef CrossControl::FrontLED* FRONTLEDHANDLE`

6.15.2 Function Documentation

6.15.2.1 EXTERN_C CCAUXDLL_API FRONTLEDHANDLE CCAUXDLL_CALLING_CONV GetFrontLED (void)

Factory function that creates instances of the FrontLED object.

Returns

FRONTLEDHANDLE to an allocated FrontLED structure. The returned handle needs to be deallocated using the FrontLED::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
FRONTLEDHANDLE pFrontLED = ::GetFrontLED();  
assert(pFrontLED);  
  
led_example(pFrontLED);  
  
pFrontLED->Release();
```

6.16 fixedIncludeFiles/Lightsensor.h File Reference

Classes

- struct [CrossControl::Lightsensor](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Lightsensor](#) * [LIGHTSENSORHANDLE](#)

Functions

- EXTERN_C CCAUXDLL_API [LIGHTSENSORHANDLE](#) CCAUXDLL_CALLING_CONV [GetLightsensor](#) (void)

6.16.1 Typedef Documentation

6.16.1.1 typedef CrossControl::Lightsensor* LIGHTSENSORHANDLE

6.16.2 Function Documentation

6.16.2.1 EXTERN_C CCAUXDLL_API LIGHTSENSORHANDLE CCAUXDLL_CALLING_CONV GetLightsensor (void)

Factory function that creates instances of the Lightsensor object.

Returns

LIGHTSENSORHANDLE to an allocated Lightsensor structure. The returned handle needs to be deallocated using the Lightsensor::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
LIGHTSENSORHANDLE pLightSensor = ::GetLightsensor();  
assert (pLightSensor);  
  
ls_example (pLightSensor);  
  
pLightSensor->Release();
```

6.17 fixedIncludeFiles/Power.h File Reference

Classes

- struct [CrossControl::Power](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Power](#) * POWERHANDLE

Functions

- EXTERN_C CCAUXDLL_API [POWERHANDLE](#) CCAUXDLL_CALLING-
_CONV [GetPower](#) (void)

6.17.1 Typedef Documentation

6.17.1.1 typedef [CrossControl::Power](#)* POWERHANDLE

6.17.2 Function Documentation

6.17.2.1 EXTERN_C CCAUXDLL_API POWERHANDLE CCAUXDLL_CALLING_CONV GetPower (void)

Factory function that creates instances of the Power object.

Returns

POWERHANDLE to an allocated Power structure. The returned handle needs to be deallocated using the Power::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
POWERHANDLE pPower = ::GetPower();  
assert (pPower);  
  
power_example (pPower);  
  
pPower->Release();
```

6.18 fixedIncludeFiles/Telematics.h File Reference

Classes

- struct [CrossControl::Telematics](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Telematics](#) * [TELEMATICSHANDLE](#)

Functions

- EXTERN_C CCAUXDLL_API [TELEMATICSHANDLE](#) CCAUXDLL_CALLING_CONV [GetTelematics](#) (void)

6.18.1 Typedef Documentation

6.18.1.1 typedef [CrossControl::Telematics](#)* [TELEMATICSHANDLE](#)

6.18.2 Function Documentation

6.18.2.1 EXTERN.C CCAUXDLL API TELEMATICSHANDLE CCAUXDLL_CALLING_CONV GetTelematics (void)

Factory function that creates instances of the Telematics object.

Returns

TELEMATICSHANDLE to an allocated Telematics structure. The returned handle needs to be deallocated using the Telematics::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
TELEMATICSHANDLE pTelematics = ::GetTelematics();
assert(pTelematics);

telematics_example(pTelematics);

pTelematics->Release();
```

6.19 fixedIncludeFiles/TouchScreen.h File Reference

Classes

- struct [CrossControl::TouchScreen](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::TouchScreen](#) * TOUCHSCREENHANDLE

Enumerations

- enum [CrossControl::TouchScreenModeSettings](#) { [CrossControl::MOUSE_NEXT_BOOT](#) = 0, [CrossControl::TOUCH_NEXT_BOOT](#) = 1, [CrossControl::MOUSE_NOW](#) = 2, [CrossControl::TOUCH_NOW](#) = 3 }
- enum [CrossControl::TSAdvancedSettingsParameter](#) { [CrossControl::TS_RIGHT_CLICK_TIME](#) = 0, [CrossControl::TS_LOW_LEVEL](#) = 1, [CrossControl::TS_UNTOUCHLEVEL](#) = 2, [CrossControl::TS_DEBOUNCE_TIME](#) = 3, [CrossControl::TS_DEBOUNCE_TIMEOUT_TIME](#) = 4, [CrossControl::TS_DOUBLECLICK_MAX_CLICK_TIME](#) = 5, [CrossControl::TS_DOUBLE_CLICK_TIME](#) = 6, [CrossControl::TS_MAX_RIGHTCLICK_DISTANCE](#) = 7, [CrossControl::TS_USE_DEJITTER](#) = 8, [CrossControl::TS_CALIBTATION_WIDTH](#) = 9, [CrossControl::TS_CALIBRATION_MEASUREMENTS](#) = 10, [CrossControl::TS_RESTORE_DEFAULT_SETTINGS](#) = 11 }

Functions

- EXTERN_C CCAUXDLL_API [TOUCHSCREENHANDLE](#) CCAUXDLL_CALLING_CONV [GetTouchScreen](#) (void)

6.19.1 Typedef Documentation

6.19.1.1 typedef [CrossControl::TouchScreen*](#) [TOUCHSCREENHANDLE](#)

6.19.2 Function Documentation

6.19.2.1 EXTERN_C CCAUXDLL_API [TOUCHSCREENHANDLE](#)
CCAUXDLL_CALLING_CONV [GetTouchScreen](#) (void)

Factory function that creates instances of the TouchScreen object.

Returns

[TOUCHSCREENHANDLE](#) to an allocated TouchScreen structure. The returned handle needs to be deallocated using the TouchScreen::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
TOUCHSCREENHANDLE pTouchScreen = ::GetTouchScreen();  
assert (pTouchScreen);  
  
touchscreen_example (pTouchScreen);  
  
pTouchScreen->Release ();
```

6.20 fixedIncludeFiles/TouchScreenCalib.h File Reference

Classes

- struct [CrossControl::TouchScreenCalib](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::TouchScreenCalib *](#) [TOUCHSCREENCALIBHANDLE](#)

Enumerations

- enum `CrossControl::CalibrationModeSettings` { `CrossControl::MODE_UNKNOWN` = 0, `CrossControl::MODE_NORMAL` = 1, `CrossControl::MODE_CALIBRATION_5P` = 2, `CrossControl::MODE_CALIBRATION_9P` = 3, `CrossControl::MODE_CALIBRATION_13P` = 4 }
- enum `CrossControl::CalibrationConfigParam` { `CrossControl::CONFIG_CALIBRATION_WITH` = 0, `CrossControl::CONFIG_CALIBRATION_MEASUREMENTS` = 1, `CrossControl::CONFIG_5P_CALIBRATION_POINT_BORDER` = 2, `CrossControl::CONFIG_13P_CALIBRATION_POINT_BORDER` = 3, `CrossControl::CONFIG_13P_CALIBRATION_TRANSITION_MIN` = 4, `CrossControl::CONFIG_13P_CALIBRATION_TRANSITION_MAX` = 5 }

Functions

- `EXTERN_C CCAUXDLL_API TOUCHSCREENCALIBHANDLE CCAUXDLL_CALLING_CONV GetTouchScreenCalib` (void)

6.20.1 Typedef Documentation

- 6.20.1.1 typedef `CrossControl::TouchScreenCalib*` `TOUCHSCREENCALIBHANDLE`

6.20.2 Function Documentation

- 6.20.2.1 `EXTERN_C CCAUXDLL_API TOUCHSCREENCALIBHANDLE CCAUXDLL_CALLING_CONV GetTouchScreenCalib` (void)

Factory function that creates instances of the `TouchScreenCalib` object.

Returns

`TOUCHSCREENCALIBHANDLE` to an allocated `TouchScreenCalib` structure. - The returned handle needs to be deallocated using the `TouchScreenCalib::Release()` method when it's no longer needed. Returns `NULL` if it fails to allocate memory.

6.21 fixedIncludeFiles/Video.h File Reference

Classes

- struct `CrossControl::Video`

Namespaces

- namespace `CrossControl`

Typedefs

- typedef [CrossControl::Video](#) * [VIDEOHANDLE](#)

Functions

- `EXTERN_C CCAUXDLL_API VIDEOHANDLE CCAUXDLL_CALLING_CONV GetVideo (void)`

6.21.1 Typedef Documentation

6.21.1.1 typedef [CrossControl::Video](#)* [VIDEOHANDLE](#)

6.21.2 Function Documentation

6.21.2.1 `EXTERN_C CCAUXDLL_API VIDEOHANDLE CCAUXDLL_CALLING_CONV GetVideo (void)`

Factory function that creates instances of the Video object.

Returns

[VIDEOHANDLE](#) to an allocated Video structure. The returned handle needs to be deallocated using the `Video::Release()` method when it's no longer needed. - Returns NULL if it fails to allocate memory.

Index

- ActionShutDown
 - CrossControl, 16
- ActionSuspend
 - CrossControl, 16
- Analog_Channel_1
 - CrossControl, 18
- Analog_Channel_2
 - CrossControl, 18
- Analog_Channel_3
 - CrossControl, 18
- Analog_Channel_4
 - CrossControl, 18
- BLUE
 - CrossControl, 10
- BPTS_BtnPressed
 - CrossControl, 8
- BPTS_BtnPressedLong
 - CrossControl, 8
- BPTS_No_Change
 - CrossControl, 8
- BPTS_Restart
 - CrossControl, 8
- BPTS_ShutDown
 - CrossControl, 8
- BPTS_SignalOff
 - CrossControl, 8
- BPTS_Suspend
 - CrossControl, 8
- Both_Button_And_Signal_Enabled
 - CrossControl, 17
- CONFIG_13P_CALIBRATION_POINT_BORDER
 - CrossControl, 9
- CONFIG_13P_CALIBRATION_TRANSITION_MAX
 - CrossControl, 9
- CONFIG_13P_CALIBRATION_TRANSITION_MIN
 - CrossControl, 9
- CONFIG_5P_CALIBRATION_POINT_BORDER
 - CrossControl, 9
- CONFIG_CALIBRATION_MEASUREMENTS
 - CrossControl, 9
- CONFIG_CALIBRATION_WITH
 - CrossControl, 9
- CYAN
 - CrossControl, 10
- CrossControl
 - ActionShutDown, 16
 - ActionSuspend, 16
 - Analog_Channel_1, 18
 - Analog_Channel_2, 18
 - Analog_Channel_3, 18
 - Analog_Channel_4, 18
 - BLUE, 10
 - BPTS_BtnPressed, 8
 - BPTS_BtnPressedLong, 8
 - BPTS_No_Change, 8
 - BPTS_Restart, 8
 - BPTS_ShutDown, 8
 - BPTS_SignalOff, 8
 - BPTS_Suspend, 8
 - Both_Button_And_Signal_Enabled, 17
 - CONFIG_13P_CALIBRATION_POINT_BORDER, 9
 - CONFIG_13P_CALIBRATION_TRANSITION_MAX, 9
 - CONFIG_13P_CALIBRATION_TRANSITION_MIN, 9
 - CONFIG_5P_CALIBRATION_POINT_BORDER, 9
 - CONFIG_CALIBRATION_MEASUREMENTS, 9
 - CONFIG_CALIBRATION_WITH, 9
 - CYAN, 10
 - DeInterlace_BOB, 10
 - DeInterlace_Even, 10
 - DeInterlace_Odd, 10
 - Disabled, 10

- ERR_AVERAGE_CALC_STARTED, 11
- ERR_BUFFER_SIZE, 10
- ERR_CHECKSUM, 11
- ERR_CODE_NOT_EXIST, 10
- ERR_COMMAND_FAILED, 11
- ERR_CREATE_THREAD, 11
- ERR_DATATYPE_MISMATCH, 10
- ERR_DEVICE_READ_DATA_FAILED, 11
- ERR_DEVICE_WRITE_DATA_FAILED, 11
- ERR_EEPROM, 11
- ERR_I2C_EXPANDER_INIT_FAILED, 11
- ERR_I2C_EXPANDER_READ_FAILED, 11
- ERR_I2C_EXPANDER_WRITE_FAILED, 11
- ERR_INIT_FAILED, 11
- ERR_INVALID_DATA, 10
- ERR_INVALID_PARAMETER, 10
- ERR_IN_PROGRESS, 11
- ERR_IOCTL_FAILED, 10
- ERR_JIDA_TEMP, 11
- ERR_MEM_ALLOC_FAIL, 11
- ERR_NEWER_FPGA_VERSION_REQUIRED, 11
- ERR_NEWER_FRONT_VERSION_REQUIRED, 11
- ERR_NEWER_SS_VERSION_REQUIRED, 11
- ERR_NOT_RUNNING, 11
- ERR_NOT_SUPPORTED, 10
- ERR_OPEN_FAILED, 10
- ERR_SUCCESS, 10
- ERR_TELEMATICS_BT_NOT_AVAILABLE, 11
- ERR_TELEMATICS_GPRS_NOT_AVAILABLE, 11
- ERR_TELEMATICS_GPS_NOT_AVAILABLE, 11
- ERR_TELEMATICS_WLAN_NOT_AVAILABLE, 11
- ERR_UNKNOWN_FEATURE, 10
- ERR_VERIFY_FAILED, 11
- Enabled, 10
- FrameExtended, 9
- FrameStandard, 9
- FrameStandardExtended, 9
- Front_Button_Enabled, 17
- GREEN, 10
- MAGENTA, 10
- MODE_CALIBRATION_13P, 9
- MODE_CALIBRATION_5P, 9
- MODE_CALIBRATION_9P, 9
- MODE_NORMAL, 9
- MODE_UNKNOWN, 9
- MOUSE_NEXT_BOOT, 17
- MOUSE_NOW, 17
- NoAction, 16
- OnOff_Signal_Enabled, 17
- RED, 10
- RangeExtended, 15
- RangeStandard, 15
- STD_B_D_G_H_I_N_PAL, 19
- STD_M_J_NTSC, 19
- STD_M_PAL, 19
- STD_NTSC, 19
- STD_PAL, 19
- STD_SECAM, 19
- SamplingModeAuto, 16
- SamplingModeExtended, 16
- SamplingModeStandard, 16
- TEMP_BACKPLANE, 15
- TEMP_BOARD, 15
- TEMP_BOX, 15
- TEMP_CHIPSETS, 15
- TEMP_CPU, 15
- TEMP_ENV, 15
- TEMP_OTHER, 15
- TEMP_VIDEO, 15
- TOUCH_NEXT_BOOT, 17
- TOUCH_NOW, 17
- TS_CALIBRATION_MEASUREMENTS, 18
- TS_CALIBTATION_WIDTH, 18
- TS_DEBOUNCE_TIME, 17
- TS_DEBOUNCE_TIMEOUT_TIME, 17
- TS_DOUBLECLICK_MAX_CLICK_TIME, 17
- TS_DOUBLE_CLICK_TIME, 17
- TS_LOW_LEVEL, 17
- TS_MAX_RIGHTCLICK_DISTANCE, 18
- TS_RESTORE_DEFAULT_SETTINGS, 18
- TS_RIGHT_CLICK_TIME, 17
- TS_UNTOUCHLEVEL, 17

- TS_USE_DEJITTER, 18
- UNDEFINED_COLOR, 10
- UPGRADE_COMPLETE, 18
- UPGRADE_COMPLETE_WITH_ERRORS, 18
- UPGRADE_CONVERTING_FILE, 18
- UPGRADE_FLASHING, 18
- UPGRADE_INIT, 18
- UPGRADE_PREP_COM, 18
- UPGRADE_READING_FILE, 18
- UPGRADE_VERIFYING, 18
- VOLTAGE_0V9, 19
- VOLTAGE_12V, 19
- VOLTAGE_12VID, 19
- VOLTAGE_1V0, 19
- VOLTAGE_1V05, 19
- VOLTAGE_1V2, 19
- VOLTAGE_1V5, 19
- VOLTAGE_1V8, 19
- VOLTAGE_1V9, 19
- VOLTAGE_24V, 19
- VOLTAGE_24VIN, 19
- VOLTAGE_3V3, 19
- VOLTAGE_5V, 19
- VOLTAGE_5VSTB, 19
- VOLTAGE_VREF_INT, 19
- VOLTAGE_VTFT, 19
- YELLOW, 10
- errCode0V9OutOfLimits, 12
- errCode0V9ReadErr, 12
- errCode12VOutOfLimits, 12
- errCode12VReadErr, 12
- errCode12VVIDEOOutOfLimits, 12
- errCode12VVIDEORReadErr, 12
- errCode1V05OutOfLimits, 12
- errCode1V05ReadErr, 12
- errCode1V0OutOfLimits, 12
- errCode1V0ReadErr, 12
- errCode1V2OutOfLimits, 12
- errCode1V2ReadErr, 12
- errCode1V5OutOfLimits, 12
- errCode1V5ReadErr, 12
- errCode1V8OutOfLimits, 12
- errCode1V8ReadErr, 12
- errCode1V9OutOfLimits, 12
- errCode1V9ReadErr, 12
- errCode24VINOutOfLimits, 12
- errCode24VINReadErr, 12
- errCode24VOutOfLimits, 12
- errCode24VReadErr, 12
- errCode3V3OutOfLimits, 12
- errCode3V3ReadErr, 12
- errCode5VOutOfLimits, 12
- errCode5VReadErr, 12
- errCode5VSTBYOutOfLimits, 12
- errCode5VSTBYReadErr, 12
- errCodeBLTYPEReadErrEEPROM, 13
- errCodeBLTYPEUnexpEEPROMVal, 13
- errCodeCANPWRENOff, 14
- errCodeCANPWRENOn, 14
- errCodeCANPWROKReadErr, 12
- errCodeCANPWRReadErrEEPROM, 13
- errCodeCANPWRUnexpEEPROMVal, 13
- errCodeCBRESETReadErr, 11
- errCodeDriverInit, 13
- errCodeEXTCTRLONReadErr, 12
- errCodeEXTCTRLReadErrEEPROM, 13
- errCodeEXTCTRLTimOutReadErrEEPROM, 15
- errCodeEXTCTRLUnexpEEPROMVal, 13
- errCodeEXTCTRLUnexpVal, 14
- errCodeEXTFANOff, 14
- errCodeEXTFANOn, 14
- errCodeEXTFANReadErrEEPROM, 13
- errCodeEXTFANUnexpEEPROMVal, 13
- errCodeFPBTNAndExtCtrlDisabled, 15
- errCodeFPBTNNonReadErr, 12
- errCodeFPBTNReadErrEEPROM, 13
- errCodeFPBNTimOutReadErrEEPROM, 14
- errCodeFPBTNUexpEEPROMVal, 13
- errCodeFPBTNUexpVal, 14
- errCodeFPGACONFReadErr, 11
- errCodeFPGACONFUnexpVal, 11
- errCodeFRONTSPAREReadErr, 14
- errCodeHEATACTOff, 14
- errCodeHEATACTOn, 14
- errCodeHeatingTempReadErrEEPROM, 15
- errCodeI2CTEMPOOutOfLimits, 13
- errCodeI2CTEMPReadErr, 13
- errCodeLEDReadErrEEPROM, 13

- errCodeLEDUnexpEEPROMVal, 13
- errCodeLVDSBLENReadErr, 12
- errCodeLVDSVDDENReadErr, 12
- errCodeLowRange24VReadErrEEPROM, 13
- errCodeLowRange24VUnexpEEPROMVal, 13
- errCodeMAINPWROKReadErr, 14
- errCodeMAX, 15
- errCodeMPFailedStart, 15
- errCodeMPPWRENOFF, 14
- errCodeMPPWRENON, 14
- errCodeManageActDeAct, 15
- errCodeManageCANPwr, 14
- errCodeManageDiagnostics, 14
- errCodeManageLED, 14
- errCodeManageMPPwr, 14
- errCodeManagePowSup, 14
- errCodeManageReset, 14
- errCodeManageVidPwr, 14
- errCodeMaxSuspTimeReadErrEEPROM, 13
- errCodeNoErr, 11
- errCodeOffBL, 14
- errCodeOnBL, 14
- errCodeOperateModeStateError, 15
- errCodePG5VSTBYReadErr, 12
- errCodePG5VSTBYUnexpVal, 12
- errCodePWRENOFF, 14
- errCodePWRENON, 14
- errCodeRCCInit, 13
- errCodeReadErrEEPROM, 15
- errCodeRelPWRBTN, 14
- errCodeRelSUPPLYRESET, 13
- errCodeRelSYSRESET, 13
- errCodeSSState, 14
- errCodeSTM32TEMPOutOfLimits, 13
- errCodeSTM32TEMPReadErr, 13
- errCodeSUS3ReadErr, 11
- errCodeSUS4ReadErr, 11
- errCodeSUS5ReadErr, 11
- errCodeSWVerReadErr, 15
- errCodeSWVerWriteErr, 15
- errCodeSetLEDCol, 14
- errCodeSetLEDFreq, 14
- errCodeSetPWRBTN, 13
- errCodeSetSUPPLYRESET, 13
- errCodeSetSYSRESET, 13
- errCodeSuspToRAMReadErrEEPROM, 13
- errCodeSuspToRAMUnexpEEPROMVal, 13
- errCodeTFTVLOOutOfLimits, 12
- errCodeTFTVOLReadErr, 12
- errCodeTIMERReadErr, 14
- errCodeTickTimeOutTimer, 15
- errCodeTimeOutWaitingForVoltages, 15
- errCodeUnitTypeReadErrEEPROM, 13
- errCodeUnitTypeUnexpEEPROMVal, 13
- errCodeVID1PWRENOFF, 14
- errCodeVID1PWRENON, 14
- errCodeVID1PWRReadErrEEPROM, 13
- errCodeVID1PWRUnexpEEPROMVal, 13
- errCodeVID2PWRENOFF, 14
- errCodeVID2PWRENON, 14
- errCodeVID2PWRReadErrEEPROM, 13
- errCodeVID2PWRUnexpEEPROMVal, 13
- errCodeVID3PWRENOFF, 14
- errCodeVID3PWRENON, 14
- errCodeVID3PWRReadErrEEPROM, 13
- errCodeVID3PWRUnexpEEPROMVal, 13
- errCodeVID4PWRENOFF, 14
- errCodeVID4PWRENON, 14
- errCodeVID4PWRReadErrEEPROM, 13
- errCodeVID4PWRUnexpEEPROMVal, 13
- errCodeVIDPWROKReadErr, 12
- errCodeVarWrapAround, 14
- shutdownReasonCodeNoError, 16
- startupReasonCodeButtonPress, 16
- startupReasonCodeExtCtrl, 16
- startupReasonCodeMPRestart, 16
- startupReasonCodePowerOnStartup, 16
- startupReasonCodeUndefined, 16
- DeInterlace_BOB
 - CrossControl, 10
- DeInterlace_Even
 - CrossControl, 10
- DeInterlace_Odd
 - CrossControl, 10
- Disabled

- CrossControl, 10
- ERR_AVERAGE_CALC_STARTED
 - CrossControl, 11
- ERR_BUFFER_SIZE
 - CrossControl, 10
- ERR_CHECKSUM
 - CrossControl, 11
- ERR_CODE_NOT_EXIST
 - CrossControl, 10
- ERR_COMMAND_FAILED
 - CrossControl, 11
- ERR_CREATE_THREAD
 - CrossControl, 11
- ERR_DATATYPE_MISMATCH
 - CrossControl, 10
- ERR_DEVICE_READ_DATA_FAILED
 - CrossControl, 11
- ERR_DEVICE_WRITE_DATA_FAILED
 - CrossControl, 11
- ERR_EEPROM
 - CrossControl, 11
- ERR_I2C_EXPANDER_INIT_FAILED
 - CrossControl, 11
- ERR_I2C_EXPANDER_READ_FAILED
 - CrossControl, 11
- ERR_I2C_EXPANDER_WRITE_FAILED
 - CrossControl, 11
- ERR_INIT_FAILED
 - CrossControl, 11
- ERR_INVALID_DATA
 - CrossControl, 10
- ERR_INVALID_PARAMETER
 - CrossControl, 10
- ERR_IN_PROGRESS
 - CrossControl, 11
- ERR_IOCTL_FAILED
 - CrossControl, 10
- ERR_JIDA_TEMP
 - CrossControl, 11
- ERR_MEM_ALLOC_FAIL
 - CrossControl, 11
- ERR_NEWER_FPGA_VERSION_REQUIRED
 - CrossControl, 11
- ERR_NEWER_FRONT_VERSION_REQUIRED
 - CrossControl, 11
- ERR_NEWER_SS_VERSION_REQUIRED
 - CrossControl, 11
- CrossControl, 11
- ERR_NOT_RUNNING
 - CrossControl, 11
- ERR_NOT_SUPPORTED
 - CrossControl, 10
- ERR_OPEN_FAILED
 - CrossControl, 10
- ERR_SUCCESS
 - CrossControl, 10
- ERR_TELEMATICS_BT_NOT_AVAILABLE
 - CrossControl, 11
- ERR_TELEMATICS_GPRS_NOT_AVAILABLE
 - CrossControl, 11
- ERR_TELEMATICS_GPS_NOT_AVAILABLE
 - CrossControl, 11
- ERR_TELEMATICS_WLAN_NOT_AVAILABLE
 - CrossControl, 11
- ERR_UNKNOWN_FEATURE
 - CrossControl, 10
- ERR_VERIFY_FAILED
 - CrossControl, 11
- Enabled
 - CrossControl, 10
- FrameExtended
 - CrossControl, 9
- FrameStandard
 - CrossControl, 9
- FrameStandardExtended
 - CrossControl, 9
- Front_Button_Enabled
 - CrossControl, 17
- GREEN
 - CrossControl, 10
- MAGENTA
 - CrossControl, 10
- MODE_CALIBRATION_13P
 - CrossControl, 9
- MODE_CALIBRATION_5P
 - CrossControl, 9
- MODE_CALIBRATION_9P
 - CrossControl, 9
- MODE_NORMAL
 - CrossControl, 9
- MODE_UNKNOWN
 - CrossControl, 9
- MOUSE_NEXT_BOOT

- CrossControl, 17
- MOUSE_NOW
 - CrossControl, 17
- NoAction
 - CrossControl, 16
- OnOff_Signal_Enabled
 - CrossControl, 17
- RED
 - CrossControl, 10
- RangeExtended
 - CrossControl, 15
- RangeStandard
 - CrossControl, 15
- STD_B_D_G_H_I_N_PAL
 - CrossControl, 19
- STD_M_J_NTSC
 - CrossControl, 19
- STD_M_PAL
 - CrossControl, 19
- STD_NTSC
 - CrossControl, 19
- STD_PAL
 - CrossControl, 19
- STD_SECAM
 - CrossControl, 19
- SamplingModeAuto
 - CrossControl, 16
- SamplingModeExtended
 - CrossControl, 16
- SamplingModeStandard
 - CrossControl, 16
- TEMP_BACKPLANE
 - CrossControl, 15
- TEMP_BOARD
 - CrossControl, 15
- TEMP_BOX
 - CrossControl, 15
- TEMP_CHIPSETS
 - CrossControl, 15
- TEMP_CPU
 - CrossControl, 15
- TEMP_ENV
 - CrossControl, 15
- TEMP_OTHER
 - CrossControl, 15
- TEMP_VIDEO
 - CrossControl, 15
- TOUCH_NEXT_BOOT
 - CrossControl, 17
- TOUCH_NOW
 - CrossControl, 17
- CrossControl, 17
- TS_CALIBRATION_MEASUREMENTS
 - CrossControl, 18
- TS_CALIBRATION_WIDTH
 - CrossControl, 18
- TS_DEBOUNCE_TIME
 - CrossControl, 17
- TS_DEBOUNCE_TIMEOUT_TIME
 - CrossControl, 17
- TS_DOUBLECLICK_MAX_CLICK_TIME
 - CrossControl, 17
- TS_DOUBLE_CLICK_TIME
 - CrossControl, 17
- TS_LOW_LEVEL
 - CrossControl, 17
- TS_MAX_RIGHTCLICK_DISTANCE
 - CrossControl, 18
- TS_RESTORE_DEFAULT_SETTINGS
 - CrossControl, 18
- TS_RIGHT_CLICK_TIME
 - CrossControl, 17
- TS_UNTOUCHLEVEL
 - CrossControl, 17
- TS_USE_DEJITTER
 - CrossControl, 18
- UNDEFINED_COLOR
 - CrossControl, 10
- UPGRADE_COMPLETE
 - CrossControl, 18
- UPGRADE_COMPLETE_WITH_ERRORS
 - CrossControl, 18
- UPGRADE_CONVERTING_FILE
 - CrossControl, 18
- UPGRADE_FLASHING
 - CrossControl, 18
- UPGRADE_INIT
 - CrossControl, 18
- UPGRADE_PREP_COM
 - CrossControl, 18
- UPGRADE_READING_FILE
 - CrossControl, 18
- UPGRADE_VERIFYING
 - CrossControl, 18
- VOLTAGE_0V9
 - CrossControl, 19
- VOLTAGE_12V
 - CrossControl, 19

- VOLTAGE_12VID
 - CrossControl, 19
- VOLTAGE_1V0
 - CrossControl, 19
- VOLTAGE_1V05
 - CrossControl, 19
- VOLTAGE_1V2
 - CrossControl, 19
- VOLTAGE_1V5
 - CrossControl, 19
- VOLTAGE_1V8
 - CrossControl, 19
- VOLTAGE_1V9
 - CrossControl, 19
- VOLTAGE_24V
 - CrossControl, 19
- VOLTAGE_24VIN
 - CrossControl, 19
- VOLTAGE_3V3
 - CrossControl, 19
- VOLTAGE_5V
 - CrossControl, 19
- VOLTAGE_5VSTB
 - CrossControl, 19
- VOLTAGE_VREF_INT
 - CrossControl, 19
- VOLTAGE_VTFT
 - CrossControl, 19
- YELLOW
 - CrossControl, 10
- ABOUTHANDLE
 - About.h, 151
- ADCHANDLE
 - Adc.h, 153
- AUXVERSIONHANDLE
 - AuxVersion.h, 154
- About.h
 - ABOUTHANDLE, 151
 - GetAbout, 151
- Above80RunTime
 - CrossControl::TimerType, 128
- Adc.h
 - ADCHANDLE, 153
 - GetAdc, 153
- AuxVersion.h
 - AUXVERSIONHANDLE, 154
 - GetAuxVersion, 154
- BACKLIGHTHANDLE
 - Backlight.h, 155
- BUZZERHANDLE
 - Buzzer.h, 156
- Backlight.h
 - BACKLIGHTHANDLE, 155
 - GetBacklight, 155
- ButtonPowerTransitionStatus
 - CrossControl, 8
- Buzzer.h
 - BUZZERHANDLE, 156
 - GetBuzzer, 156
- CANSETTINGHANDLE
 - CanSetting.h, 157
- CCAuxColor
 - CrossControl, 9
- CCStatus
 - CrossControl, 10
- CONFIGHANDLE
 - Config.h, 160
- CalibrationConfigParam
 - CrossControl, 8
- CalibrationModeSettings
 - CrossControl, 9
- CanFrameType
 - CrossControl, 9
- CanSetting.h
 - CANSETTINGHANDLE, 157
 - GetCanSetting, 157
- CheckCalibrationPointFinished
 - CrossControl::TouchScreenCalib, 134
- Config.h
 - CONFIGHANDLE, 160
 - GetConfig, 160
- CrossControl, 4
 - ButtonPowerTransitionStatus, 8
 - CCAuxColor, 9
 - CCStatus, 10
 - CalibrationConfigParam, 8
 - CalibrationModeSettings, 9
 - CanFrameType, 9
 - DeInterlaceMode, 10
 - DigitalIn_1, 21
 - DigitalIn_2, 21
 - DigitalIn_3, 21
 - DigitalIn_4, 21
 - GetErrorStringA, 19
 - GetErrorStringW, 20
 - GetHwErrorStatusStringA, 20
 - GetHwErrorStatusStringW, 20
 - GetStartupReasonStringA, 21
 - GetStartupReasonStringW, 21
 - JidaSensorType, 15

- LightSensorOperationRange, 15
- LightSensorSamplingMode, 15
- PowerAction, 16
- TSAdvancedSettingsParameter, 17
- TouchScreenModeSettings, 16
- TriggerConf, 17
- UpgradeAction, 18
- VersionType, 8
- Video1Conf, 21
- Video2Conf, 21
- Video3Conf, 22
- Video4Conf, 22
- VideoChannel, 18
- VoltageEnum, 19
- eErr, 10
- hwErrorStatusCodes, 11
- shutDownReasonCodes, 16
- startupReasonCodes, 16
- videoStandard, 18
- CrossControl::About, 23
 - Release, 36
 - getAddOnHWversion, 26
 - getAddOnManufacturingDate, 26
 - getAddOnPCBArt, 27
 - getAddOnPCBSerial, 27
 - getIsBTMounted, 28
 - getIsDisplayAvailable, 28
 - getIsGPRSMounted, 28
 - getIsGPSPMounted, 29
 - getIsTouchScreenAvailable, 29
 - getIsWLANMounted, 30
 - getMainHWversion, 30
 - getMainManufacturingDate, 31
 - getMainPCBArt, 31
 - getMainPCBSerial, 31
 - getMainProdArtNr, 32
 - getMainProdRev, 32
 - getNrOfCANConnections, 33
 - getNrOfDigIOConnections, 33
 - getNrOfETHConnections, 34
 - getNrOfSerialConnections, 34
 - getNrOfUSBConnections, 34
 - getNrOfVideoConnections, 35
 - getUnitSerial, 35
- CrossControl::Adc, 36
 - Release, 38
 - getVoltage, 37
- CrossControl::AuxVersion, 38
 - Release, 45
 - getCCAuxDrvVersion, 41
 - getCCAuxVersion, 42
 - getFPGAVersion, 42
 - getFrontVersion, 43
 - getOSVersion, 44
 - getSSVersion, 44
- CrossControl::Backlight, 45
 - Release, 50
 - getAutomaticBLFilter, 47
 - getAutomaticBLParams, 48
 - getAutomaticBLStatus, 48
 - getIntensity, 48
 - getLedDimming, 49
 - getStatus, 49
 - setAutomaticBLFilter, 50
 - setAutomaticBLParams, 51
 - setIntensity, 51
 - setLedDimming, 51
 - startAutomaticBL, 52
 - stopAutomaticBL, 52
- CrossControl::Buzzer, 52
 - Release, 56
 - buzze, 54
 - getFrequency, 55
 - getTrigger, 55
 - getVolume, 56
 - setFrequency, 56
 - setTrigger, 57
 - setVolume, 57
- CrossControl::BuzzerSetup, 58
 - frequency, 58
 - volume, 58
- CrossControl::CanSetting, 58
 - Release, 61
 - getBaudrate, 60
 - getFrameType, 60
 - setBaudrate, 61
 - setFrameType, 62
- CrossControl::Config, 62
 - Release, 69
 - getCanStartupPowerConfig, 64
 - getExtFanStartupPowerConfig, 64
 - getExtOnOffSigTrigTime, 65
 - getFrontBtnTrigTime, 65
 - getHeatingTempLimit, 65
 - getLongButtonPressAction, 66
 - getOnOffSigAction, 66
 - getPowerOnStartup, 66
 - getShortButtonPressAction, 67
 - getStartupTriggerConfig, 67
 - getStartupVoltageConfig, 68

- getSuspendMaxTime, 68
- getVideoStartupPowerConfig, 68
- setCanStartupPowerConfig, 69
- setExtFanStartupPowerConfig, 69
- setExtOnOffSigTrigTime, 70
- setFrontBtnTrigTime, 70
- setHeatingTempLimit, 70
- setLongButtonPressAction, 71
- setOnOffSigAction, 71
- setPowerOnStartup, 71
- setShortButtonPressAction, 72
- setStartupTriggerConfig, 72
- setStartupVoltageConfig, 72
- setSuspendMaxTime, 73
- setVideoStartupPowerConfig, 73
- CrossControl::Diagnostic, 74
 - Release, 79
 - clearHwErrorStatus, 76
 - getHwErrorStatus, 76
 - getMinMaxTemp, 76
 - getPCBTemp, 76
 - getPMTemp, 77
 - getPowerCycles, 77
 - getSSTemp, 78
 - getShutDownReason, 77
 - getStartupReason, 78
 - getTimer, 79
- CrossControl::DigIO, 79
 - Release, 82
 - getDigIO, 81
 - getDigIODir, 81
 - setDigIO, 82
 - setDigIODir, 82
- CrossControl::FirmwareUpgrade, 83
 - Release, 86
 - getUpgradeStatus, 86
 - shutDown, 87
 - startFpgaUpgrade, 87
 - startFpgaVerification, 88
 - startFrontUpgrade, 89
 - startFrontVerification, 90
 - startSSUpgrade, 91
 - startSSVerification, 92
- CrossControl::FrontLED, 93
 - Release, 98
 - getColor, 95, 96
 - getEnabledDuringStartup, 96
 - getIdleTime, 96
 - getNrOfPulses, 97
 - getOffTime, 97
 - getOnTime, 97
 - getSignal, 98
 - setColor, 98, 99
 - getEnabledDuringStartup, 99
 - getIdleTime, 100
 - setNrOfPulses, 100
 - setOff, 100
 - setOffTime, 100
 - setOnTime, 101
 - setSignal, 101
- CrossControl::LedColorMixType, 102
 - blue, 102
 - green, 102
 - red, 102
- CrossControl::LedTimingType, 102
 - idleTime, 103
 - nrOfPulses, 103
 - offTime, 103
 - onTime, 103
- CrossControl::Lightsensor, 103
 - Release, 106
 - getAverageIlluminance, 105
 - getIlluminance, 105, 106
 - getOperatingRange, 106
 - setOperatingRange, 107
 - startAverageCalc, 107
 - stopAverageCalc, 108
- CrossControl::Power, 108
 - Release, 112
 - ackPowerRequest, 110
 - getBLPowerStatus, 110
 - getButtonPowerTransitionStatus, 111
 - getCanPowerStatus, 111
 - getExtFanPowerStatus, 111
 - getVideoPowerStatus, 112
 - setBLPowerStatus, 113
 - setCanPowerStatus, 113
 - setExtFanPowerStatus, 114
 - setVideoPowerStatus, 114
- CrossControl::Telematics, 115
 - Release, 125
 - getBTPowerStatus, 119
 - getBTStartUpPowerStatus, 119
 - getGPRSPowerStatus, 120
 - getGPRSStartUpPowerStatus, 120
 - getGPSAntennaStatus, 121
 - getGPSPowerStatus, 122
 - getGPSStartUpPowerStatus, 122
 - getTelematicsAvailable, 123
 - getWLANPowerStatus, 123

- getWLANStartUpPowerStatus, 124
- setBTPowerStatus, 125
- setBTStartUpPowerStatus, 125
- setGPRSPowerStatus, 126
- setGPRSStartUpPowerStatus, 126
- setGPSPowerStatus, 126
- setGPSStartUpPowerStatus, 127
- setWLANPowerStatus, 127
- setWLANStartUpPowerStatus, 127
- CrossControl::TimerType, 128
 - Above80RunTime, 128
 - RunTime40_60, 128
 - RunTime60_70, 128
 - RunTime70_80, 128
 - TotHeatTime, 128
 - TotRunTime, 128
 - TotSuspTime, 129
- CrossControl::TouchScreen, 129
 - Release, 132
 - getAdvancedSetting, 130
 - getMode, 131
 - getMouseRightClickTime, 132
 - setAdvancedSetting, 133
 - setMode, 133
 - setMouseRightClickTime, 133
- CrossControl::TouchScreenCalib, 134
 - CheckCalibrationPointFinished, 134
 - GetConfigParam, 135
 - GetMode, 135
 - Release, 135
 - SetCalibrationPoint, 135
 - SetConfigParam, 136
 - SetMode, 136
- CrossControl::UpgradeStatus, 136
 - currentAction, 137
 - errorCode, 137
 - percent, 137
- CrossControl::Video, 138
 - Release, 144
 - activateSnapshot, 139
 - createBitmap, 139
 - freeBmpBuffer, 140
 - getActiveChannel, 140
 - getColorKeys, 141
 - getCropping, 141
 - getDeInterlaceMode, 142
 - getDecoderReg, 141
 - getMirroring, 142
 - getRawImage, 142
 - getScaling, 143
 - getStatus, 143
 - getVideoArea, 143
 - getVideoStandard, 144
 - init, 144
 - minimize, 144
 - restore, 145
 - setActiveChannel, 145
 - setColorKeys, 145
 - setCropping, 145
 - setDeInterlaceMode, 146
 - setDecoderReg, 146
 - setMirroring, 147
 - setScaling, 147
 - setVideoArea, 147
 - showVideo, 148
 - takeSnapshot, 148
 - takeSnapshotBmp, 148
 - takeSnapshotRaw, 149
- CrossControl::received_video, 114
 - received_framerate, 115
 - received_height, 115
 - received_width, 115
- CrossControl::version_info, 137
 - build, 137
 - major, 137
 - minor, 137
 - release, 138
- CrossControl::video_dec_command, 149
 - decoder_register, 150
 - register_value, 150
- DIAGNOSTICHANDLE
 - Diagnostic.h, 161
- DIGIOHANDLE
 - DigIO.h, 165
- DeInterlaceMode
 - CrossControl, 10
- Diagnostic.h
 - DIAGNOSTICHANDLE, 161
 - GetDiagnostic, 161
- DigIO.h
 - DIGIOHANDLE, 165
 - GetDigIO, 165
- DigitalIn_1
 - CrossControl, 21
- DigitalIn_2
 - CrossControl, 21
- DigitalIn_3
 - CrossControl, 21
- DigitalIn_4
 - CrossControl, 21

- FIRMWAREUPGHANDLE
 - FirmwareUpgrade.h, 166
- FRONTLEDHANDLE
 - FrontLED.h, 166
- FirmwareUpgrade.h
 - FIRMWAREUPGHANDLE, 166
 - GetFirmwareUpgrade, 166
- FrontLED.h
 - FRONTLEDHANDLE, 166
 - GetFrontLED, 167
- GetAbout
 - About.h, 151
- GetAdc
 - Adc.h, 153
- GetAuxVersion
 - AuxVersion.h, 154
- GetBacklight
 - Backlight.h, 155
- GetBuzzer
 - Buzzer.h, 156
- GetCanSetting
 - CanSetting.h, 157
- GetConfig
 - Config.h, 160
- GetConfigParam
 - CrossControl::TouchScreenCalib, 135
- GetDiagnostic
 - Diagnostic.h, 161
- GetDigIO
 - DigIO.h, 165
- GetErrorStringA
 - CrossControl, 19
- GetErrorStringW
 - CrossControl, 20
- GetFirmwareUpgrade
 - FirmwareUpgrade.h, 166
- GetFrontLED
 - FrontLED.h, 167
- GetHwErrorStatusStringA
 - CrossControl, 20
- GetHwErrorStatusStringW
 - CrossControl, 20
- GetLightsensor
 - Lightsensor.h, 167
- GetMode
 - CrossControl::TouchScreenCalib, 135
- GetPower
 - Power.h, 168
- GetStartupReasonStringA
 - CrossControl, 21
- GetStartupReasonStringW
 - CrossControl, 21
- GetTelematics
 - Telematics.h, 169
- GetTouchScreen
 - TouchScreen.h, 171
- GetTouchScreenCalib
 - TouchScreenCalib.h, 172
- GetVideo
 - Video.h, 173
- JidaSensorType
 - CrossControl, 15
- LIGHTSENSORHANDLE
 - Lightsensor.h, 167
- LightSensorOperationRange
 - CrossControl, 15
- LightSensorSamplingMode
 - CrossControl, 15
- Lightsensor.h
 - GetLightsensor, 167
 - LIGHTSENSORHANDLE, 167
- POWERHANDLE
 - Power.h, 168
- Power.h
 - GetPower, 168
 - POWERHANDLE, 168
- PowerAction
 - CrossControl, 16
- Release
 - CrossControl::About, 36
 - CrossControl::Adc, 38
 - CrossControl::AuxVersion, 45
 - CrossControl::Backlight, 50
 - CrossControl::Buzzer, 56
 - CrossControl::CanSetting, 61
 - CrossControl::Config, 69
 - CrossControl::Diagnostic, 79
 - CrossControl::DigIO, 82
 - CrossControl::FirmwareUpgrade, 86
 - CrossControl::FrontLED, 98
 - CrossControl::Lightsensor, 106
 - CrossControl::Power, 112
 - CrossControl::Telematics, 125
 - CrossControl::TouchScreen, 132
 - CrossControl::TouchScreenCalib, 135
 - CrossControl::Video, 144
- RunTime40_60
 - CrossControl::TimerType, 128
- RunTime60_70
 - CrossControl::TimerType, 128

- RunTime70_80
 - CrossControl::TimerType, 128
- SetCalibrationPoint
 - CrossControl::TouchScreenCalib, 135
- SetConfigParam
 - CrossControl::TouchScreenCalib, 136
- SetMode
 - CrossControl::TouchScreenCalib, 136
- TELEMATICSHANDLE
 - Telematics.h, 169
- TOUCHSCREENHANDLE
 - TouchScreen.h, 171
- TSAdvancedSettingsParameter
 - CrossControl, 17
- Telematics.h
 - GetTelematics, 169
 - TELEMATICSHANDLE, 169
- TotHeatTime
 - CrossControl::TimerType, 128
- TotRunTime
 - CrossControl::TimerType, 128
- TotSuspTime
 - CrossControl::TimerType, 129
- TouchScreen.h
 - GetTouchScreen, 171
 - TOUCHSCREENHANDLE, 171
- TouchScreenCalib.h
 - GetTouchScreenCalib, 172
- TouchScreenModeSettings
 - CrossControl, 16
- TriggerConf
 - CrossControl, 17
- UpgradeAction
 - CrossControl, 18
- VIDEOHANDLE
 - Video.h, 173
- VersionType
 - CrossControl, 8
- Video.h
 - GetVideo, 173
 - VIDEOHANDLE, 173
- Video1Conf
 - CrossControl, 21
- Video2Conf
 - CrossControl, 21
- Video3Conf
 - CrossControl, 22
- Video4Conf
 - CrossControl, 22
- VideoChannel
 - CrossControl, 18
- VoltageEnum
 - CrossControl, 19
- ackPowerRequest
 - CrossControl::Power, 110
- activateSnapshot
 - CrossControl::Video, 139
- blue
 - CrossControl::LedColorMixType, 102
- build
 - CrossControl::version_info, 137
- buzze
 - CrossControl::Buzzer, 54
- clearHwErrorStatus
 - CrossControl::Diagnostic, 76
- createBitmap
 - CrossControl::Video, 139
- currentAction
 - CrossControl::UpgradeStatus, 137
- decoder_register
 - CrossControl::video_dec_command, 150
- eErr
 - CrossControl, 10
- errCode0V9OutOfLimits
 - CrossControl, 12
- errCode0V9ReadErr
 - CrossControl, 12
- errCode12VOutOfLimits
 - CrossControl, 12
- errCode12VReadErr
 - CrossControl, 12
- errCode12VVIDEOOutOfLimits
 - CrossControl, 12
- errCode12VVIDEORedErr
 - CrossControl, 12
- errCode1V05OutOfLimits
 - CrossControl, 12
- errCode1V05ReadErr
 - CrossControl, 12
- errCode1V0OutOfLimits
 - CrossControl, 12
- errCode1V0ReadErr
 - CrossControl, 12
- errCode1V2OutOfLimits
 - CrossControl, 12
- errCode1V2ReadErr
 - CrossControl, 12

- CrossControl, [12](#)
- errCode1V5OutOfLimits
 - CrossControl, [12](#)
- errCode1V5ReadErr
 - CrossControl, [12](#)
- errCode1V8OutOfLimits
 - CrossControl, [12](#)
- errCode1V8ReadErr
 - CrossControl, [12](#)
- errCode1V9OutOfLimits
 - CrossControl, [12](#)
- errCode1V9ReadErr
 - CrossControl, [12](#)
- errCode24VINOutOfLimits
 - CrossControl, [12](#)
- errCode24VINReadErr
 - CrossControl, [12](#)
- errCode24VOutOfLimits
 - CrossControl, [12](#)
- errCode24VReadErr
 - CrossControl, [12](#)
- errCode3V3OutOfLimits
 - CrossControl, [12](#)
- errCode3V3ReadErr
 - CrossControl, [12](#)
- errCode5VOutOfLimits
 - CrossControl, [12](#)
- errCode5VReadErr
 - CrossControl, [12](#)
- errCode5VSTBYOutOfLimits
 - CrossControl, [12](#)
- errCode5VSTBYReadErr
 - CrossControl, [12](#)
- errCodeBLTYPEReadErrEEPROM
 - CrossControl, [13](#)
- errCodeBLTYPEUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeCANPWRENOff
 - CrossControl, [14](#)
- errCodeCANPWRENOn
 - CrossControl, [14](#)
- errCodeCANPWROKReadErr
 - CrossControl, [12](#)
- errCodeCANPWRReadErrEEPROM
 - CrossControl, [13](#)
- errCodeCANPWRUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeCBRESETReadErr
 - CrossControl, [11](#)
- errCodeDriverInit
 - CrossControl, [13](#)
- errCodeEXTCTRLONReadErr
 - CrossControl, [12](#)
- errCodeEXTCTRLReadErrEEPROM
 - CrossControl, [13](#)
- errCodeEXTCTRLTimOutReadErrEEPROM
 - CrossControl, [15](#)
- errCodeEXTCTRLUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeEXTCTRLUnexpVal
 - CrossControl, [14](#)
- errCodeEXTFANOff
 - CrossControl, [14](#)
- errCodeEXTFANOn
 - CrossControl, [14](#)
- errCodeEXTFANReadErrEEPROM
 - CrossControl, [13](#)
- errCodeEXTFANUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeFPBTNAndExtCtrlDisabled
 - CrossControl, [15](#)
- errCodeFPBTNONReadErr
 - CrossControl, [12](#)
- errCodeFPBTNReadErrEEPROM
 - CrossControl, [13](#)
- errCodeFPBTNTimOutReadErrEEPROM
 - CrossControl, [14](#)
- errCodeFPBTNUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeFPBTNUnexpVal
 - CrossControl, [14](#)
- errCodeFPGACONFReadErr
 - CrossControl, [11](#)
- errCodeFPGACONFUnexpVal
 - CrossControl, [11](#)
- errCodeFRONTSPAREReadErr
 - CrossControl, [14](#)
- errCodeHEATACTOff
 - CrossControl, [14](#)
- errCodeHEATACTOn
 - CrossControl, [14](#)
- errCodeHeatingTempReadErrEEPROM
 - CrossControl, [15](#)
- errCodeI2CTEMPOutOfLimits
 - CrossControl, [13](#)
- errCodeI2CTEMPReadErr
 - CrossControl, [13](#)
- errCodeLEDReadErrEEPROM
 - CrossControl, [13](#)

- errCodeLEDUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeLVDSBLENReadErr
 - CrossControl, [12](#)
- errCodeLVDSVDDENReadErr
 - CrossControl, [12](#)
- errCodeLowRange24VReadErrEEPROM
 - CrossControl, [13](#)
- errCodeLowRange24VUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeMAINPWOKReadErr
 - CrossControl, [14](#)
- errCodeMAX
 - CrossControl, [15](#)
- errCodeMPFailedStart
 - CrossControl, [15](#)
- errCodeMPPWRENOff
 - CrossControl, [14](#)
- errCodeMPPWRENOn
 - CrossControl, [14](#)
- errCodeManageActDeAct
 - CrossControl, [15](#)
- errCodeManageCANPwr
 - CrossControl, [14](#)
- errCodeManageDiagnostics
 - CrossControl, [14](#)
- errCodeManageLED
 - CrossControl, [14](#)
- errCodeManageMPPwr
 - CrossControl, [14](#)
- errCodeManagePowSup
 - CrossControl, [14](#)
- errCodeManageReset
 - CrossControl, [14](#)
- errCodeManageVidPwr
 - CrossControl, [14](#)
- errCodeMaxSuspTimeReadErrEEPROM
 - CrossControl, [13](#)
- errCodeNoErr
 - CrossControl, [11](#)
- errCodeOffBL
 - CrossControl, [14](#)
- errCodeOnBL
 - CrossControl, [14](#)
- errCodeOperateModeStateError
 - CrossControl, [15](#)
- errCodePG5VSTBYReadErr
 - CrossControl, [12](#)
- errCodePG5VSTBYUnexpVal
 - CrossControl, [12](#)
- errCodePWRENOff
 - CrossControl, [14](#)
- errCodePWRENOn
 - CrossControl, [14](#)
- errCodeRCCInit
 - CrossControl, [13](#)
- errCodeReadErrEEPROM
 - CrossControl, [15](#)
- errCodeRelPWRBTN
 - CrossControl, [14](#)
- errCodeRelSUPPLYRESET
 - CrossControl, [13](#)
- errCodeRelSYSRESET
 - CrossControl, [13](#)
- errCodeSSState
 - CrossControl, [14](#)
- errCodeSTM32TEMPOutOfLimits
 - CrossControl, [13](#)
- errCodeSTM32TEMPReadErr
 - CrossControl, [13](#)
- errCodeSUS3ReadErr
 - CrossControl, [11](#)
- errCodeSUS4ReadErr
 - CrossControl, [11](#)
- errCodeSUS5ReadErr
 - CrossControl, [11](#)
- errCodeSWVerReadErr
 - CrossControl, [15](#)
- errCodeSWVerWriteErr
 - CrossControl, [15](#)
- errCodeSetLEDCol
 - CrossControl, [14](#)
- errCodeSetLEDFreq
 - CrossControl, [14](#)
- errCodeSetPWRBTN
 - CrossControl, [13](#)
- errCodeSetSUPPLYRESET
 - CrossControl, [13](#)
- errCodeSetSYSRESET
 - CrossControl, [13](#)
- errCodeSuspToRAMReadErrEEPROM
 - CrossControl, [13](#)
- errCodeSuspToRAMUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeTFTVOLOutOfLimits
 - CrossControl, [12](#)
- errCodeTFTVOLReadErr
 - CrossControl, [12](#)
- errCodeTIMERReadErr
 - CrossControl, [14](#)

- errCodeTickTimeOutTimer
 - CrossControl, [15](#)
- errCodeTimeOutWaitingForVoltages
 - CrossControl, [15](#)
- errCodeUnitTypeReadErrEEPROM
 - CrossControl, [13](#)
- errCodeUnitTypeUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeVID1PWRENOff
 - CrossControl, [14](#)
- errCodeVID1PWRENOOn
 - CrossControl, [14](#)
- errCodeVID1PWRReadErrEEPROM
 - CrossControl, [13](#)
- errCodeVID1PWRUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeVID2PWRENOff
 - CrossControl, [14](#)
- errCodeVID2PWRENOOn
 - CrossControl, [14](#)
- errCodeVID2PWRReadErrEEPROM
 - CrossControl, [13](#)
- errCodeVID2PWRUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeVID3PWRENOff
 - CrossControl, [14](#)
- errCodeVID3PWRENOOn
 - CrossControl, [14](#)
- errCodeVID3PWRReadErrEEPROM
 - CrossControl, [13](#)
- errCodeVID3PWRUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeVID4PWRENOff
 - CrossControl, [14](#)
- errCodeVID4PWRENOOn
 - CrossControl, [14](#)
- errCodeVID4PWRReadErrEEPROM
 - CrossControl, [13](#)
- errCodeVID4PWRUnexpEEPROMVal
 - CrossControl, [13](#)
- errCodeVIDPWROKReadErr
 - CrossControl, [12](#)
- errCodeVarWrapAround
 - CrossControl, [14](#)
- errorCode
 - CrossControl::UpgradeStatus, [137](#)
- fixedIncludeFiles/About.h, [151](#)
- fixedIncludeFiles/Adc.h, [152](#)
- fixedIncludeFiles/AuxVersion.h, [153](#)
- fixedIncludeFiles/Backlight.h, [154](#)
- fixedIncludeFiles/Buzzer.h, [155](#)
- fixedIncludeFiles/CCAuxErrors.h, [157](#)
- fixedIncludeFiles/CCAuxTypes.h, [158](#)
- fixedIncludeFiles/CCPlatform.h, [159](#)
- fixedIncludeFiles/CanSetting.h, [156](#)
- fixedIncludeFiles/Config.h, [160](#)
- fixedIncludeFiles/Diagnostic.h, [161](#)
- fixedIncludeFiles/DiagnosticCodes.h, [162](#)
- fixedIncludeFiles/DigIO.h, [164](#)
- fixedIncludeFiles/FirmwareUpgrade.h, [165](#)
- fixedIncludeFiles/FrontLED.h, [166](#)
- fixedIncludeFiles/Lightsensor.h, [167](#)
- fixedIncludeFiles/Power.h, [168](#)
- fixedIncludeFiles/Telematics.h, [169](#)
- fixedIncludeFiles/TouchScreen.h, [170](#)
- fixedIncludeFiles/TouchScreenCalib.h, [171](#)
- fixedIncludeFiles/Video.h, [172](#)
- freeBmpBuffer
 - CrossControl::Video, [140](#)
- frequency
 - CrossControl::BuzzerSetup, [58](#)
- getActiveChannel
 - CrossControl::Video, [140](#)
- getAddOnHWversion
 - CrossControl::About, [26](#)
- getAddOnManufacturingDate
 - CrossControl::About, [26](#)
- getAddOnPCBArt
 - CrossControl::About, [27](#)
- getAddOnPCBSerial
 - CrossControl::About, [27](#)
- getAdvancedSetting
 - CrossControl::TouchScreen, [130](#)
- getAutomaticBLFilter
 - CrossControl::Backlight, [47](#)
- getAutomaticBLParams
 - CrossControl::Backlight, [48](#)
- getAutomaticBLStatus
 - CrossControl::Backlight, [48](#)
- getAverageIlluminance
 - CrossControl::Lightsensor, [105](#)
- getBLPowerStatus
 - CrossControl::Power, [110](#)
- getBTPowerStatus
 - CrossControl::Telematics, [119](#)
- getBTStartUpPowerStatus
 - CrossControl::Telematics, [119](#)
- getBaudrate

- CrossControl::CanSetting, 60
- getButtonPowerTransitionStatus
 - CrossControl::Power, 111
- getCCAuxDrvVersion
 - CrossControl::AuxVersion, 41
- getCCAuxVersion
 - CrossControl::AuxVersion, 42
- getCanPowerStatus
 - CrossControl::Power, 111
- getCanStartupPowerConfig
 - CrossControl::Config, 64
- getColor
 - CrossControl::FrontLED, 95, 96
- getColorKeys
 - CrossControl::Video, 141
- getCropping
 - CrossControl::Video, 141
- getDeInterlaceMode
 - CrossControl::Video, 142
- getDecoderReg
 - CrossControl::Video, 141
- getDigIO
 - CrossControl::DigIO, 81
- getDigIODir
 - CrossControl::DigIO, 81
- getEnabledDuringStartup
 - CrossControl::FrontLED, 96
- getExtFanPowerStatus
 - CrossControl::Power, 111
- getExtFanStartupPowerConfig
 - CrossControl::Config, 64
- getExtOnOffSigTrigTime
 - CrossControl::Config, 65
- getFPGAVersion
 - CrossControl::AuxVersion, 42
- getFrameType
 - CrossControl::CanSetting, 60
- getFrequency
 - CrossControl::Buzzer, 55
- getFrontBtnTrigTime
 - CrossControl::Config, 65
- getFrontVersion
 - CrossControl::AuxVersion, 43
- getGPRSPowerStatus
 - CrossControl::Telematics, 120
- getGPRSStartupPowerStatus
 - CrossControl::Telematics, 120
- getGPSAntennaStatus
 - CrossControl::Telematics, 121
- getGPSPowerStatus
 - CrossControl::Telematics, 122
- getGPSStartupPowerStatus
 - CrossControl::Telematics, 122
- getHeatingTempLimit
 - CrossControl::Config, 65
- getHwErrorStatus
 - CrossControl::Diagnostic, 76
- getIdleTime
 - CrossControl::FrontLED, 96
- getIlluminance
 - CrossControl::Lightsensor, 105, 106
- getIntensity
 - CrossControl::Backlight, 48
- getIsBTMounted
 - CrossControl::About, 28
- getIsDisplayAvailable
 - CrossControl::About, 28
- getIsGPRSMounted
 - CrossControl::About, 28
- getIsGPSMounted
 - CrossControl::About, 29
- getIsTouchScreenAvailable
 - CrossControl::About, 29
- getIsWLANMounted
 - CrossControl::About, 30
- getLedDimming
 - CrossControl::Backlight, 49
- getLongButtonPressAction
 - CrossControl::Config, 66
- getMainHWversion
 - CrossControl::About, 30
- getMainManufacturingDate
 - CrossControl::About, 31
- getMainPCBArt
 - CrossControl::About, 31
- getMainPCBSerial
 - CrossControl::About, 31
- getMainProdArtNr
 - CrossControl::About, 32
- getMainProdRev
 - CrossControl::About, 32
- getMinMaxTemp
 - CrossControl::Diagnostic, 76
- getMirroring
 - CrossControl::Video, 142
- getMode
 - CrossControl::TouchScreen, 131
- getMouseRightClickTime
 - CrossControl::TouchScreen, 132
- getNrOfCANConnections

- CrossControl::About, 33
- getNrOfDigIOConnections
 - CrossControl::About, 33
- getNrOfETHConnections
 - CrossControl::About, 34
- getNrOfPulses
 - CrossControl::FrontLED, 97
- getNrOfSerialConnections
 - CrossControl::About, 34
- getNrOfUSBConnections
 - CrossControl::About, 34
- getNrOfVideoConnections
 - CrossControl::About, 35
- getOSVersion
 - CrossControl::AuxVersion, 44
- getOffTime
 - CrossControl::FrontLED, 97
- getOnOffSigAction
 - CrossControl::Config, 66
- getOnTime
 - CrossControl::FrontLED, 97
- getOperatingRange
 - CrossControl::Lightsensor, 106
- getPCBTemp
 - CrossControl::Diagnostic, 76
- getPMTemp
 - CrossControl::Diagnostic, 77
- getPowerCycles
 - CrossControl::Diagnostic, 77
- getPowerOnStartup
 - CrossControl::Config, 66
- getRawImage
 - CrossControl::Video, 142
- getSSTemp
 - CrossControl::Diagnostic, 78
- getSSVersion
 - CrossControl::AuxVersion, 44
- getScaling
 - CrossControl::Video, 143
- getShortButtonPressAction
 - CrossControl::Config, 67
- getShutDownReason
 - CrossControl::Diagnostic, 77
- getSignal
 - CrossControl::FrontLED, 98
- getStartupReason
 - CrossControl::Diagnostic, 78
- getStartupTriggerConfig
 - CrossControl::Config, 67
- getStartupVoltageConfig
 - CrossControl::Config, 68
- getStatus
 - CrossControl::Backlight, 49
 - CrossControl::Video, 143
- getSuspendMaxTime
 - CrossControl::Config, 68
- getTelematicsAvailable
 - CrossControl::Telematics, 123
- getTimer
 - CrossControl::Diagnostic, 79
- getTrigger
 - CrossControl::Buzzer, 55
- getUnitSerial
 - CrossControl::About, 35
- getUpgradeStatus
 - CrossControl::FirmwareUpgrade, 86
- getVideoArea
 - CrossControl::Video, 143
- getVideoPowerStatus
 - CrossControl::Power, 112
- getVideoStandard
 - CrossControl::Video, 144
- getVideoStartupPowerConfig
 - CrossControl::Config, 68
- getVoltage
 - CrossControl::Adc, 37
- getVolume
 - CrossControl::Buzzer, 56
- getWLANPowerStatus
 - CrossControl::Telematics, 123
- getWLANStartUpPowerStatus
 - CrossControl::Telematics, 124
- green
 - CrossControl::LedColorMixType, 102
- hwErrorStatusCodes
 - CrossControl, 11
- idleTime
 - CrossControl::LedTimingType, 103
- init
 - CrossControl::Video, 144
- major
 - CrossControl::version_info, 137
- minimize
 - CrossControl::Video, 144
- minor
 - CrossControl::version_info, 137
- nrOfPulses

- CrossControl::LedTimingType, 103
- offTime
 - CrossControl::LedTimingType, 103
- onTime
 - CrossControl::LedTimingType, 103
- percent
 - CrossControl::UpgradeStatus, 137
- received_framerate
 - CrossControl::received_video, 115
- received_height
 - CrossControl::received_video, 115
- received_width
 - CrossControl::received_video, 115
- red
 - CrossControl::LedColorMixType, 102
- register_value
 - CrossControl::video_dec_command, 150
- release
 - CrossControl::version_info, 138
- restore
 - CrossControl::Video, 145
- setActiveChannel
 - CrossControl::Video, 145
- setAdvancedSetting
 - CrossControl::TouchScreen, 133
- setAutomaticBLFilter
 - CrossControl::Backlight, 50
- setAutomaticBLParams
 - CrossControl::Backlight, 51
- setBLPowerStatus
 - CrossControl::Power, 113
- setBTPowerStatus
 - CrossControl::Telematics, 125
- setBTStartupPowerStatus
 - CrossControl::Telematics, 125
- setBaudrate
 - CrossControl::CanSetting, 61
- setCanPowerStatus
 - CrossControl::Power, 113
- setCanStartupPowerConfig
 - CrossControl::Config, 69
- setColor
 - CrossControl::FrontLED, 98, 99
- setColorKeys
 - CrossControl::Video, 145
- setCropping
 - CrossControl::Video, 145
- setDeInterlaceMode
 - CrossControl::Video, 146
- setDecoderReg
 - CrossControl::Video, 146
- setDigIO
 - CrossControl::DigIO, 82
- setDigIODir
 - CrossControl::DigIO, 82
- setEnabledDuringStartup
 - CrossControl::FrontLED, 99
- setExtFanPowerStatus
 - CrossControl::Power, 114
- setExtFanStartupPowerConfig
 - CrossControl::Config, 69
- setExtOnOffSigTrigTime
 - CrossControl::Config, 70
- setFrameType
 - CrossControl::CanSetting, 62
- setFrequency
 - CrossControl::Buzzer, 56
- setFrontBtnTrigTime
 - CrossControl::Config, 70
- setGPRSPowerStatus
 - CrossControl::Telematics, 126
- setGPRSStartUpPowerStatus
 - CrossControl::Telematics, 126
- setGPSPowerStatus
 - CrossControl::Telematics, 126
- setGPSStartUpPowerStatus
 - CrossControl::Telematics, 127
- setHeatingTempLimit
 - CrossControl::Config, 70
- setIdleTime
 - CrossControl::FrontLED, 100
- setIntensity
 - CrossControl::Backlight, 51
- setLedDimming
 - CrossControl::Backlight, 51
- setLongButtonPressAction
 - CrossControl::Config, 71
- setMirroring
 - CrossControl::Video, 147
- setMode
 - CrossControl::TouchScreen, 133
- setMouseRightClickTime
 - CrossControl::TouchScreen, 133
- setNrOfPulses
 - CrossControl::FrontLED, 100
- setOff

- CrossControl::FrontLED, [100](#)
- setOffTime
 - CrossControl::FrontLED, [100](#)
- setOnOffSigAction
 - CrossControl::Config, [71](#)
- setOnTime
 - CrossControl::FrontLED, [101](#)
- setOperatingRange
 - CrossControl::Lightsensor, [107](#)
- setPowerOnStartup
 - CrossControl::Config, [71](#)
- setScaling
 - CrossControl::Video, [147](#)
- setShortButtonPressAction
 - CrossControl::Config, [72](#)
- setSignal
 - CrossControl::FrontLED, [101](#)
- setStartupTriggerConfig
 - CrossControl::Config, [72](#)
- setStartupVoltageConfig
 - CrossControl::Config, [72](#)
- setSuspendMaxTime
 - CrossControl::Config, [73](#)
- setTrigger
 - CrossControl::Buzzer, [57](#)
- setVideoArea
 - CrossControl::Video, [147](#)
- setVideoPowerStatus
 - CrossControl::Power, [114](#)
- setVideoStartupPowerConfig
 - CrossControl::Config, [73](#)
- setVolume
 - CrossControl::Buzzer, [57](#)
- setWLANPowerStatus
 - CrossControl::Telematics, [127](#)
- setWLANStartUpPowerStatus
 - CrossControl::Telematics, [127](#)
- showVideo
 - CrossControl::Video, [148](#)
- shutDown
 - CrossControl::FirmwareUpgrade, [87](#)
- shutDownReasonCodes
 - CrossControl, [16](#)
- shutdownReasonCodeNoError
 - CrossControl, [16](#)
- startAutomaticBL
 - CrossControl::Backlight, [52](#)
- startAverageCalc
 - CrossControl::Lightsensor, [107](#)
- startFpgaUpgrade
 - CrossControl::FirmwareUpgrade, [87](#)
- startFpgaVerification
 - CrossControl::FirmwareUpgrade, [88](#)
- startFrontUpgrade
 - CrossControl::FirmwareUpgrade, [89](#)
- startFrontVerification
 - CrossControl::FirmwareUpgrade, [90](#)
- startSSUpgrade
 - CrossControl::FirmwareUpgrade, [91](#)
- startSSVerification
 - CrossControl::FirmwareUpgrade, [92](#)
- startupReasonCodeButtonPress
 - CrossControl, [16](#)
- startupReasonCodeExtCtrl
 - CrossControl, [16](#)
- startupReasonCodeMPRestart
 - CrossControl, [16](#)
- startupReasonCodePowerOnStartup
 - CrossControl, [16](#)
- startupReasonCodeUndefined
 - CrossControl, [16](#)
- startupReasonCodes
 - CrossControl, [16](#)
- stopAutomaticBL
 - CrossControl::Backlight, [52](#)
- stopAverageCalc
 - CrossControl::Lightsensor, [108](#)
- takeSnapshot
 - CrossControl::Video, [148](#)
- takeSnapshotBmp
 - CrossControl::Video, [148](#)
- takeSnapshotRaw
 - CrossControl::Video, [149](#)
- videoStandard
 - CrossControl, [18](#)
- volume
 - CrossControl::BuzzerSetup, [58](#)