

cr•ssc•ntr•l

CCAux
1.3.1.0

Sat Feb 18 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	CanFrameType	8
4.1.2.3	CCAuxColor	9
4.1.2.4	CCStatus	9
4.1.2.5	DeInterlaceMode	9
4.1.2.6	eErr	9
4.1.2.7	hwErrorStatusCodes	10
4.1.2.8	JidaSensorType	14
4.1.2.9	LightSensorOperationRange	14
4.1.2.10	LightSensorSamplingMode	15
4.1.2.11	PowerAction	15
4.1.2.12	shutDownReasonCodes	15
4.1.2.13	startupReasonCodes	15
4.1.2.14	TouchScreenModeSettings	16
4.1.2.15	TriggerConf	16
4.1.2.16	TSAdvancedSettingsParameter	16
4.1.2.17	UpgradeAction	17
4.1.2.18	VideoChannel	17
4.1.2.19	videoStandard	18
4.1.2.20	VoltageEnum	18
4.1.3	Function Documentation	18
4.1.3.1	GetErrorStringA	18
4.1.3.2	GetErrorStringW	19
4.1.3.3	GetHwErrorStatusStringA	19
4.1.3.4	GetHwErrorStatusStringW	19

4.1.3.5	GetStartupReasonStringA	20
4.1.3.6	GetStartupReasonStringW	20
4.1.4	Variable Documentation	20
4.1.4.1	DigitalIn_1	20
4.1.4.2	DigitalIn_2	20
4.1.4.3	DigitalIn_3	20
4.1.4.4	DigitalIn_4	20
4.1.4.5	Video1Conf	20
4.1.4.6	Video2Conf	21
4.1.4.7	Video3Conf	21
4.1.4.8	Video4Conf	21
5	Class Documentation	22
5.1	CrossControl::About Struct Reference	22
5.1.1	Detailed Description	23
5.1.2	Member Function Documentation	25
5.1.2.1	getAddOnHWversion	25
5.1.2.2	getAddOnManufacturingDate	25
5.1.2.3	getAddOnPCBArt	26
5.1.2.4	getAddOnPCBSerial	26
5.1.2.5	getIsBTMounted	26
5.1.2.6	getIsGPRSMounted	27
5.1.2.7	getIsGPSMounted	27
5.1.2.8	getIsWLANMounted	28
5.1.2.9	getMainHWversion	28
5.1.2.10	getMainManufacturingDate	29
5.1.2.11	getMainPCBArt	29
5.1.2.12	getMainPCBSerial	29
5.1.2.13	getMainProdArtNr	30
5.1.2.14	getMainProdRev	30
5.1.2.15	getNrOfCANConnections	31
5.1.2.16	getNrOfETHConnections	31
5.1.2.17	getNrOfSerialConnections	32
5.1.2.18	getNrOfUSBConnections	32
5.1.2.19	getNrOfVideoConnections	32
5.1.2.20	getUnitSerial	33
5.1.2.21	Release	33
5.2	CrossControl::Adc Struct Reference	34
5.2.1	Detailed Description	34
5.2.2	Member Function Documentation	35
5.2.2.1	getVoltage	35
5.2.2.2	Release	35
5.3	CrossControl::AuxVersion Struct Reference	36
5.3.1	Detailed Description	36
5.3.2	Member Function Documentation	38
5.3.2.1	getCCAuxDrvVersion	39
5.3.2.2	getCCAuxVersion	39
5.3.2.3	getFPGAVersion	40
5.3.2.4	getFrontVersion	41
5.3.2.5	getOSVersion	41

5.3.2.6	getSSVersion	42
5.3.2.7	Release	42
5.4	CrossControl::Backlight Struct Reference	43
5.4.1	Detailed Description	43
5.4.2	Member Function Documentation	43
5.4.2.1	getAutomaticBLFilter	43
5.4.2.2	getAutomaticBLParams	44
5.4.2.3	getAutomaticBLStatus	44
5.4.2.4	getIntensity	44
5.4.2.5	getLedDimming	45
5.4.2.6	getStatus	45
5.4.2.7	Release	45
5.4.2.8	setAutomaticBLFilter	46
5.4.2.9	setAutomaticBLParams	46
5.4.2.10	setIntensity	46
5.4.2.11	setLedDimming	47
5.4.2.12	startAutomaticBL	47
5.4.2.13	stopAutomaticBL	47
5.5	CrossControl::Buzzer Struct Reference	47
5.5.1	Detailed Description	48
5.5.2	Member Function Documentation	48
5.5.2.1	buzze	48
5.5.2.2	getFrequency	48
5.5.2.3	getTrigger	49
5.5.2.4	getVolume	49
5.5.2.5	Release	49
5.5.2.6	setFrequency	49
5.5.2.7	setTrigger	50
5.5.2.8	setVolume	50
5.6	CrossControl::BuzzerSetup Struct Reference	50
5.6.1	Member Data Documentation	50
5.6.1.1	frequency	50
5.6.1.2	volume	51
5.7	CrossControl::CanSetting Struct Reference	51
5.7.1	Detailed Description	51
5.7.2	Member Function Documentation	51
5.7.2.1	getBaudrate	51
5.7.2.2	getFrameType	51
5.7.2.3	Release	52
5.7.2.4	setBaudrate	52
5.7.2.5	setFrameType	52
5.8	CrossControl::Config Struct Reference	53
5.8.1	Detailed Description	53
5.8.2	Member Function Documentation	54
5.8.2.1	getCanStartupPowerConfig	54
5.8.2.2	getExtFanStartupPowerConfig	54
5.8.2.3	getExtOnOffSigTrigTime	54
5.8.2.4	getFrontBtnTrigTime	55
5.8.2.5	getHeatingTempLimit	55
5.8.2.6	getLongButtonPressAction	55

5.8.2.7	getOnOffSigAction	56
5.8.2.8	getPowerOnStartup	56
5.8.2.9	getShortButtonPressAction	56
5.8.2.10	getStartupTriggerConfig	57
5.8.2.11	getStartupVoltageConfig	57
5.8.2.12	getSuspendMaxTime	57
5.8.2.13	getVideoStartupPowerConfig	58
5.8.2.14	Release	58
5.8.2.15	setCanStartupPowerConfig	58
5.8.2.16	setExtFanStartupPowerConfig	58
5.8.2.17	setExtOnOffSigTrigTime	59
5.8.2.18	setFrontBtnTrigTime	59
5.8.2.19	setHeatingTempLimit	59
5.8.2.20	setLongButtonPressAction	60
5.8.2.21	setOnOffSigAction	60
5.8.2.22	setPowerOnStartup	60
5.8.2.23	setShortButtonPressAction	61
5.8.2.24	setStartupTriggerConfig	61
5.8.2.25	setStartupVoltageConfig	61
5.8.2.26	setSuspendMaxTime	62
5.8.2.27	setVideoStartupPowerConfig	62
5.9	CrossControl::Diagnostic Struct Reference	62
5.9.1	Detailed Description	63
5.9.2	Member Function Documentation	63
5.9.2.1	clearHwErrorStatus	63
5.9.2.2	getHwErrorStatus	63
5.9.2.3	getMinMaxTemp	64
5.9.2.4	getPCBTemp	64
5.9.2.5	getPMTemp	64
5.9.2.6	getPowerCycles	65
5.9.2.7	getShutDownReason	65
5.9.2.8	getSSTemp	65
5.9.2.9	getStartupReason	65
5.9.2.10	getTimer	66
5.9.2.11	Release	66
5.10	CrossControl::DigIO Struct Reference	66
5.10.1	Detailed Description	67
5.10.2	Member Function Documentation	67
5.10.2.1	getDigIO	67
5.10.2.2	Release	68
5.11	CrossControl::FirmwareUpgrade Struct Reference	68
5.11.1	Detailed Description	69
5.11.2	Member Function Documentation	69
5.11.2.1	getUpgradeStatus	69
5.11.2.2	Release	69
5.11.2.3	shutDown	70
5.11.2.4	startFpgaUpgrade	70
5.11.2.5	startFpgaVerification	71
5.11.2.6	startFrontUpgrade	72
5.11.2.7	startFrontVerification	73

5.11.2.8	startSSUpgrade	74
5.11.2.9	startSSVerification	75
5.12	CrossControl::FrontLED Struct Reference	76
5.12.1	Detailed Description	76
5.12.2	Member Function Documentation	77
5.12.2.1	getColor	77
5.12.2.2	getColor	77
5.12.2.3	getEnabledDuringStartup	77
5.12.2.4	getIdleTime	78
5.12.2.5	getNrOfPulses	78
5.12.2.6	getOffTime	78
5.12.2.7	getOnTime	79
5.12.2.8	getSignal	79
5.12.2.9	Release	79
5.12.2.10	setColor	79
5.12.2.11	setColor	80
5.12.2.12	setEnabledDuringStartup	80
5.12.2.13	setIdleTime	80
5.12.2.14	setNrOfPulses	81
5.12.2.15	setOff	81
5.12.2.16	setOffTime	81
5.12.2.17	setOnTime	81
5.12.2.18	setSignal	82
5.13	CrossControl::LedColorMixType Struct Reference	82
5.13.1	Member Data Documentation	82
5.13.1.1	blue	82
5.13.1.2	green	82
5.13.1.3	red	83
5.14	CrossControl::LedTimingType Struct Reference	83
5.14.1	Member Data Documentation	83
5.14.1.1	idleTime	83
5.14.1.2	nrOfPulses	83
5.14.1.3	offTime	83
5.14.1.4	onTime	83
5.15	CrossControl::Lightsensor Struct Reference	83
5.15.1	Detailed Description	84
5.15.2	Member Function Documentation	84
5.15.2.1	getAverageIlluminance	84
5.15.2.2	getIlluminance	84
5.15.2.3	getIlluminance	85
5.15.2.4	getOperatingRange	85
5.15.2.5	Release	85
5.15.2.6	setOperatingRange	85
5.15.2.7	startAverageCalc	86
5.15.2.8	stopAverageCalc	86
5.16	CrossControl::Power Struct Reference	86
5.16.1	Detailed Description	87
5.16.2	Member Function Documentation	87
5.16.2.1	ackPowerRequest	87
5.16.2.2	getBLPowerStatus	87

5.16.2.3	getButtonPowerTransitionStatus	88
5.16.2.4	getCanPowerStatus	88
5.16.2.5	getExtFanPowerStatus	88
5.16.2.6	getVideoPowerStatus	88
5.16.2.7	Release	89
5.16.2.8	setBLPowerStatus	89
5.16.2.9	setCanPowerStatus	89
5.16.2.10	setExtFanPowerStatus	90
5.16.2.11	setVideoPowerStatus	90
5.17	CrossControl::received_video Struct Reference	90
5.17.1	Member Data Documentation	91
5.17.1.1	received_framerate	91
5.17.1.2	received_height	91
5.17.1.3	received_width	91
5.18	CrossControl::Telematics Struct Reference	91
5.18.1	Detailed Description	91
5.18.2	Member Function Documentation	91
5.18.2.1	getGPRSPowerStatus	91
5.18.2.2	getGPRSStartupPowerStatus	92
5.18.2.3	getGPSAntennaStatus	92
5.18.2.4	getTelematicsAvailable	92
5.18.2.5	getWLANPowerStatus	93
5.18.2.6	getWLANStartupPowerStatus	93
5.18.2.7	Release	93
5.18.2.8	setGPRSPowerStatus	93
5.18.2.9	setGPRSStartupPowerStatus	94
5.18.2.10	setWLANPowerStatus	94
5.18.2.11	setWLANStartupPowerStatus	94
5.19	CrossControl::TimerType Struct Reference	95
5.19.1	Detailed Description	95
5.19.2	Member Data Documentation	95
5.19.2.1	Above80RunTime	95
5.19.2.2	RunTime40_60	95
5.19.2.3	RunTime60_70	95
5.19.2.4	RunTime70_80	95
5.19.2.5	TotHeatTime	95
5.19.2.6	TotRunTime	96
5.19.2.7	TotSuspTime	96
5.20	CrossControl::TouchScreen Struct Reference	96
5.20.1	Detailed Description	96
5.20.2	Member Function Documentation	96
5.20.2.1	getAdvancedSetting	96
5.20.2.2	getMode	97
5.20.2.3	getMouseRightClickTime	97
5.20.2.4	Release	97
5.20.2.5	setAdvancedSetting	97
5.20.2.6	setMode	98
5.20.2.7	setMouseRightClickTime	98
5.21	CrossControl::UpgradeStatus Struct Reference	98
5.21.1	Detailed Description	99

5.21.2	Member Data Documentation	99
5.21.2.1	currentAction	99
5.21.2.2	errorCode	99
5.21.2.3	percent	99
5.22	CrossControl::version_info Struct Reference	99
5.22.1	Member Data Documentation	99
5.22.1.1	build	99
5.22.1.2	major	100
5.22.1.3	minor	100
5.22.1.4	release	100
5.23	CrossControl::Video Struct Reference	100
5.23.1	Detailed Description	101
5.23.2	Member Function Documentation	101
5.23.2.1	activateSnapshot	101
5.23.2.2	createBitmap	102
5.23.2.3	freeBmpBuffer	102
5.23.2.4	getActiveChannel	102
5.23.2.5	getColorKeys	103
5.23.2.6	getCropping	103
5.23.2.7	getDecoderReg	103
5.23.2.8	getDeInterlaceMode	104
5.23.2.9	getMirroring	104
5.23.2.10	getRawImage	104
5.23.2.11	getScaling	105
5.23.2.12	getStatus	105
5.23.2.13	getVideoArea	106
5.23.2.14	getVideoStandard	106
5.23.2.15	init	106
5.23.2.16	minimize	107
5.23.2.17	Release	107
5.23.2.18	restore	107
5.23.2.19	setActiveChannel	107
5.23.2.20	setColorKeys	108
5.23.2.21	setCropping	108
5.23.2.22	setDecoderReg	108
5.23.2.23	setDeInterlaceMode	109
5.23.2.24	setMirroring	109
5.23.2.25	setScaling	109
5.23.2.26	setVideoArea	110
5.23.2.27	showVideo	110
5.23.2.28	takeSnapshot	111
5.23.2.29	takeSnapshotBmp	111
5.23.2.30	takeSnapshotRaw	112
5.24	CrossControl::video_dec_command Struct Reference	112
5.24.1	Member Data Documentation	112
5.24.1.1	decoder_register	112
5.24.1.2	register_value	112
6	File Documentation	113
6.1	fixedIncludeFiles/About.h File Reference	113

6.1.1	Typedef Documentation	113
6.1.1.1	ABOUTHANDLE	113
6.1.2	Function Documentation	113
6.1.2.1	GetAbout	113
6.2	fixedIncludeFiles/Adc.h File Reference	114
6.2.1	Typedef Documentation	115
6.2.1.1	ADCHANDLE	115
6.2.2	Function Documentation	115
6.2.2.1	GetAdc	115
6.3	fixedIncludeFiles/AuxVersion.h File Reference	115
6.3.1	Typedef Documentation	116
6.3.1.1	AUXVERSIONHANDLE	116
6.3.2	Function Documentation	116
6.3.2.1	GetAuxVersion	116
6.4	fixedIncludeFiles/Backlight.h File Reference	116
6.4.1	Typedef Documentation	117
6.4.1.1	BACKLIGHTHANDLE	117
6.4.2	Function Documentation	117
6.4.2.1	GetBacklight	117
6.5	fixedIncludeFiles/Buzzer.h File Reference	117
6.5.1	Typedef Documentation	117
6.5.1.1	BUZZERHANDLE	117
6.5.2	Function Documentation	117
6.5.2.1	GetBuzzer	118
6.6	fixedIncludeFiles/CanSetting.h File Reference	118
6.6.1	Typedef Documentation	118
6.6.1.1	CANSETTINGHANDLE	118
6.6.2	Function Documentation	118
6.6.2.1	GetCanSetting	118
6.7	fixedIncludeFiles/CCAuxErrors.h File Reference	118
6.8	fixedIncludeFiles/CCAuxTypes.h File Reference	119
6.9	fixedIncludeFiles/CCPlatform.h File Reference	120
6.10	fixedIncludeFiles/Config.h File Reference	121
6.10.1	Typedef Documentation	121
6.10.1.1	CONFIGHANDLE	121
6.10.2	Function Documentation	121
6.10.2.1	GetConfig	121
6.11	fixedIncludeFiles/Diagnostic.h File Reference	121
6.11.1	Typedef Documentation	122
6.11.1.1	DIAGNOSTICHANDLE	122
6.11.2	Function Documentation	122
6.11.2.1	GetDiagnostic	122
6.12	fixedIncludeFiles/DiagnosticCodes.h File Reference	122
6.13	fixedIncludeFiles/DigIO.h File Reference	124
6.13.1	Typedef Documentation	125
6.13.1.1	DIGIOHANDLE	125
6.13.2	Function Documentation	125
6.13.2.1	GetDigIO	125
6.14	fixedIncludeFiles/FirmwareUpgrade.h File Reference	125
6.14.1	Typedef Documentation	126

6.14.1.1	FIRMWAREUPGHANDLE	126
6.14.2	Function Documentation	126
6.14.2.1	GetFirmwareUpgrade	126
6.15	fixedIncludeFiles/FrontLED.h File Reference	126
6.15.1	Typedef Documentation	127
6.15.1.1	FRONTLEDHANDLE	127
6.15.2	Function Documentation	127
6.15.2.1	GetFrontLED	127
6.16	fixedIncludeFiles/Lightsensor.h File Reference	127
6.16.1	Typedef Documentation	127
6.16.1.1	LIGHTSENSORHANDLE	127
6.16.2	Function Documentation	127
6.16.2.1	GetLightsensor	127
6.17	fixedIncludeFiles/Power.h File Reference	127
6.17.1	Typedef Documentation	128
6.17.1.1	POWERHANDLE	128
6.17.2	Function Documentation	128
6.17.2.1	GetPower	128
6.18	fixedIncludeFiles/Telematics.h File Reference	128
6.18.1	Typedef Documentation	129
6.18.1.1	TELEMATICSHANDLE	129
6.18.2	Function Documentation	129
6.18.2.1	GetTelematics	129
6.19	fixedIncludeFiles/TouchScreen.h File Reference	129
6.19.1	Typedef Documentation	130
6.19.1.1	TOUCHSCREENHANDLE	130
6.19.2	Function Documentation	130
6.19.2.1	GetTouchScreen	130
6.20	fixedIncludeFiles/Video.h File Reference	130
6.20.1	Typedef Documentation	130
6.20.1.1	VIDEOHANDLE	130
6.20.2	Function Documentation	130
6.20.2.1	GetVideo	130

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	22
CrossControl::Adc	34
CrossControl::AuxVersion	36
CrossControl::Backlight	43
CrossControl::Buzzer	47
CrossControl::BuzzerSetup	50
CrossControl::CanSetting	51
CrossControl::Config	53
CrossControl::Diagnostic	62
CrossControl::DigIO	66
CrossControl::FirmwareUpgrade	68
CrossControl::FrontLED	76
CrossControl::LedColorMixType	82
CrossControl::LedTimingType	83
CrossControl::Lightsensor	83
CrossControl::Power	86
CrossControl::received_video	90
CrossControl::Telematics	91
CrossControl::TimerType	95
CrossControl::TouchScreen	96
CrossControl::UpgradeStatus	98
CrossControl::version_info	99
CrossControl::Video	100
CrossControl::video_dec_command	112

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

fixedIncludeFiles/About.h	113
fixedIncludeFiles/Adc.h	114
fixedIncludeFiles/AuxVersion.h	115
fixedIncludeFiles/Backlight.h	116
fixedIncludeFiles/Buzzer.h	117
fixedIncludeFiles/CanSetting.h	118
fixedIncludeFiles/CCAuxErrors.h	118
fixedIncludeFiles/CCAuxTypes.h	119
fixedIncludeFiles/CCPlatform.h	120
fixedIncludeFiles/Config.h	121
fixedIncludeFiles/Diagnostic.h	121
fixedIncludeFiles/DiagnosticCodes.h	122
fixedIncludeFiles/DigIO.h	124
fixedIncludeFiles/FirmwareUpgrade.h	125
fixedIncludeFiles/FrontLED.h	126
fixedIncludeFiles/Lightsensor.h	127
fixedIncludeFiles/Power.h	127
fixedIncludeFiles/Telematics.h	128
fixedIncludeFiles/TouchScreen.h	129
fixedIncludeFiles/Video.h	130

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 [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, errCodeCANPWOKReadErr = 9, errCodeVIDPWOKReadErr = 10, errCodeLVDSBLENReadErr = 11, errCodeLVDSVDDENReadErr = 12, errCodeEXTCTRLONReadErr = 13, errCodeFPBTNONReadErr = 14, errCode24VReadErr = 15, errCode24VOutOfLimits = 16, errCode24VINReadErr = 17, errCode24VINOutOfLimits = 18, errCode12VReadErr = 19, errCode12VOutOfLimits = 20, errCode12VVIDEORReadErr = 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, errCodeBLTYPERReadErrEEPROM = 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, [errCodeManageMPPwr](#) = 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 }

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::CanFrameType

Can frame type settings

Enumerator:

FrameStandard

FrameExtended

FrameStandardExtended

4.1.2.3 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

4.1.2.4 enum CrossControl::CCStatus

Enable/disable enumeration

Enumerator:

Disabled

Enabled The setting is disabled or turned off

4.1.2.5 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

4.1.2.6 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

4.1.2.7 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

errCodeIV0ReadErr
errCodeIV0OutOfLimits
errCodeOV9ReadErr
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

4.1.2.8 enum CrossControl::JidaSensorType

Jida temperature sensor types

Enumerator:

TEMP_CPU
TEMP_BOX
TEMP_ENV
TEMP_BOARD
TEMP_BACKPLANE
TEMP_CHIPSETS
TEMP_VIDEO
TEMP_OTHER

4.1.2.9 enum CrossControl::LightSensorOperationRange

Light sensor operation ranges.

Enumerator:

RangeStandard
RangeExtended Light sensor operation range standard

4.1.2.10 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.

4.1.2.11 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

4.1.2.12 enum CrossControl::shutDownReasonCodes

The shutdown codes returned by getShutDownReason.

Enumerator:

shutdownReasonCodeNoError

4.1.2.13 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

4.1.2.14 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.

4.1.2.15 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

4.1.2.16 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.

4.1.2.17 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

4.1.2.18 enum CrossControl::VideoChannel

The available analog video channels

Enumerator:

Analog_Channel_1

Analog_Channel_2

Analog_Channel_3

Analog_Channel_4

4.1.2.19 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

4.1.2.20 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

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;

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::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;
```

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

```

5.1.2.8 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;

```

5.1.2.9 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;

```

5.1.2.10 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;
```

5.1.2.11 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;
```

5.1.2.12 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;
```

5.1.2.13 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;
```

5.1.2.14 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;
```

5.1.2.15 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;
```

5.1.2.16 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;
```

5.1.2.17 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;
```

5.1.2.18 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;
```

5.1.2.19 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;
```

5.1.2.20 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;
```

5.1.2.21 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::eErr 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::eErr 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

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.

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.

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

Delete the backlight object.

Returns

-

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>bSoft-Transitions</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.

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

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.

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.

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

Delete the [Buzzer](#) object.

Returns

-

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.

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.

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 setFrameType](#) (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

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.

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.

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

Delete the [CanSetting](#) object.

Returns

-

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

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.

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

-

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.

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 VideoXConf 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

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

<code>errorCode</code>	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.

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.

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.

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

Delete the [Diagnostic](#) object.

Returns

-

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

Delete the [DigIO](#) object.

Returns

-

Example Usage:

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

list_digital_inputs(pDigIO);

pDigIO->Release();
```

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

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

-

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 [Firmware-Upgrade](#) 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 <code>getUpgradeStatus</code> 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 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;
}
}
}

```

5.11.2.7 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 [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 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;
        }
    }
}

```

5.11.2.8 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 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;
}
}
}

```

5.11.2.9 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 [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 <code>getUpgradeStatus</code> 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;
    }
}
}

```

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

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.

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.

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

Delete the [FrontLED](#) object.

Returns

-

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.

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.

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.

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.

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.

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 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).

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.

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>	Illuminance value (Lux).
--------------	--------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

-

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.

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

Stop average calculation.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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.

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.

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

Delete the [Power](#) object.

Returns

-

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.

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

5.18.1 Detailed Description

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

5.18.2 Member Function Documentation

5.18.2.1 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.

5.18.2.2 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.

5.18.2.3 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.

5.18.2.4 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.

5.18.2.5 `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.

5.18.2.6 `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.

5.18.2.7 `virtual void CrossControl::Telematics::Release () [pure virtual]`

Delete the [Telematics](#) object.

Returns

-

5.18.2.8 `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.

5.18.2.9 `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.

5.18.2.10 `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.

5.18.2.11 `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

Get/Set touch screen USB profile settings

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.

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.

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.

5.20.2.4 `virtual void CrossControl::TouchScreen::Release () [pure virtual]`

Delete the [TouchScreen](#) object.

Returns

-

5.20.2.5 `virtual eErr CrossControl::TouchScreen::setAdvancedSetting (TSAAdvancedSettingsParameter param, unsigned short data) [pure virtual]`

Set advanced touch screen settings. See the description of TSAAdvancedSettingsParameter 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::UpgradeStatus Struct Reference

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

Public Attributes

- enum [UpgradeAction](#) `currentAction`
- unsigned char `percent`
- [eErr](#) `errorCode`

5.21.1 Detailed Description

Upgrade Status

5.21.2 Member Data Documentation

5.21.2.1 enum [UpgradeAction](#) `CrossControl::UpgradeStatus::currentAction`

5.21.2.2 [eErr](#) `CrossControl::UpgradeStatus::errorCode`

Represents the percentage of completion of the current action

5.21.2.3 unsigned char `CrossControl::UpgradeStatus::percent`

The current action.

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

- `fixedIncludeFiles/CCAuxTypes.h`

5.22 CrossControl::version_info Struct Reference

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

Public Attributes

- unsigned char `major`
- unsigned char `minor`
- unsigned char `release`
- unsigned char `build`

5.22.1 Member Data Documentation

5.22.1.1 unsigned char `CrossControl::version_info::build`

version build number

5.22.1.2 unsigned char CrossControl::version_info::major

version major number

5.22.1.3 unsigned char CrossControl::version_info::minor

version minor number

5.22.1.4 unsigned char CrossControl::version_info::release

version release number

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

- [fixedIncludeFiles/CCAuxTypes.h](#)

5.23 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 rawImgBufSize, 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

5.23.1 Detailed Description

Analog [Video](#)

5.23.2 Member Function Documentation

5.23.2.1 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.

5.23.2.2 `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.

5.23.2.3 `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.

5.23.2.4 `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.

5.23.2.5 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.

5.23.2.6 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.

5.23.2.7 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.

5.23.2.8 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.

5.23.2.9 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.

5.23.2.10 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.

5.23.2.11 `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.

5.23.2.12 `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.

5.23.2.13 `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.

5.23.2.14 `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.

5.23.2.15 `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.

5.23.2.16 virtual eErr CrossControl::Video::minimize () [pure virtual]

Minimizes the video area. Restore with [restore\(\)](#) call.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

5.23.2.17 virtual void CrossControl::Video::Release () [pure virtual]

Delete the [Video](#) object.

Returns

-

5.23.2.18 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.

5.23.2.19 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.

5.23.2.20 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.

5.23.2.21 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.

5.23.2.22 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.

5.23.2.23 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.

5.23.2.24 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.

5.23.2.25 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.

5.23.2.26 `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.

5.23.2.27 `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.

5.23.2.28 `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.

5.23.2.29 `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.

5.23.2.30 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](#)

5.24 CrossControl::video_dec_command Struct Reference

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

Public Attributes

- unsigned char [decoder_register](#)
- unsigned char [register_value](#)

5.24.1 Member Data Documentation

5.24.1.1 unsigned char [CrossControl::video_dec_command::decoder_register](#)

5.24.1.2 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

`ADCHANDLE` 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 struct [Backlight](#) * [BACKLIGHTHANDLE](#)

6.4.2 Function Documentation

- 6.4.2.1 [EXTERN_C](#) [CCAUXDLL_API](#) [BACKLIGHTHANDLE](#) [CCAUXDLL_CALLING_CONV](#) [GetBacklight](#) (void)

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)

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)

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_GPS_NOT_AVAILABLE = 30, 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)

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)

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::errCodeLVD-SVDDENReadErr](#) = 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::errCode12VV-
IDEOOutOfLimits = 22, CrossControl::errCode5VSTBYReadErr = 23, Cross-
Control::errCode5VSTBYOutOfLimits = 24, CrossControl::errCode5VReadErr
= 25, CrossControl::errCode5VOutOfLimits = 26, CrossControl::errCode3V3-
ReadErr = 27, CrossControl::errCode3V3OutOfLimits = 28, CrossControl::err-
CodeTFTVOLReadErr = 29, CrossControl::errCodeTFTVLOOutOfLimits = 30,
CrossControl::errCode1V9ReadErr = 31, CrossControl::errCode1V9OutOfLimits
= 32, CrossControl::errCode1V8ReadErr = 33, CrossControl::errCode1V8Out-
OfLimits = 34, CrossControl::errCode1V5ReadErr = 35, CrossControl::err-
Code1V5OutOfLimits = 36, CrossControl::errCode1V2ReadErr = 37, Cross-
Control::errCode1V2OutOfLimits = 38, CrossControl::errCode1V05ReadErr =
39, CrossControl::errCode1V05OutOfLimits = 40, CrossControl::errCode1V0-
ReadErr = 41, CrossControl::errCode1V00OutOfLimits = 42, CrossControl::err-
Code0V9ReadErr = 43, CrossControl::errCode0V9OutOfLimits = 44, Cross-
Control::errCodeI2CTEMPReadErr = 45, CrossControl::errCodeI2CTEMPOut-
OfLimits = 46, CrossControl::errCodeSTM32TEMPReadErr = 47, CrossControl-
::errCodeSTM32TEMPOOutOfLimits = 48, CrossControl::errCodeBLTYPEUnexp-
EEPROMVal = 49, CrossControl::errCodeFPBTNUnexpEEPROMVal = 50, -
CrossControl::errCodeEXTCTRLUnexpEEPROMVal = 51, CrossControl::err-
CodeLowRange24VUnexpEEPROMVal = 52, CrossControl::errCodeSuspToR-
AMUnexpEEPROMVal = 53, CrossControl::errCodeCANPWRUnexpEEPRO-
MVal = 54, CrossControl::errCodeVID1PWRUnexpEEPROMVal = 55, Cross-
Control::errCodeVID2PWRUnexpEEPROMVal = 56, CrossControl::errCodeV-
ID3PWRUnexpEEPROMVal = 57, CrossControl::errCodeVID4PWRUnexpEE-
PROMVal = 58, CrossControl::errCodeEXTFANUnexpEEPROMVal = 59, -
CrossControl::errCodeLEDUnexpEEPROMVal = 60, CrossControl::errCodeUnit-
TypeUnexpEEPROMVal = 61, CrossControl::errCodeBLTYPEReadErrEEPR-
OM = 62, CrossControl::errCodeFPBTNReadErrEEPROM = 63, CrossControl-
::errCodeEXTCTRLReadErrEEPROM = 64, CrossControl::errCodeMaxSusp-
TimeReadErrEEPROM = 65, CrossControl::errCodeLowRange24VReadErrE-
EPROM = 66, CrossControl::errCodeSuspToRAMReadErrEEPROM = 67, -
CrossControl::errCodeCANPWRReadErrEEPROM = 68, CrossControl::errCode-
VID1PWRReadErrEEPROM = 69, CrossControl::errCodeVID2PWRReadErrE-
EPROM = 70, CrossControl::errCodeVID3PWRReadErrEEPROM = 71, -
CrossControl::errCodeVID4PWRReadErrEEPROM = 72, CrossControl::errCode-
EXTFANReadErrEEPROM = 73, CrossControl::errCodeLEDReadErrEEPRO-
M = 74, CrossControl::errCodeUnitTypeReadErrEEPROM = 75, CrossControl-
::errCodeRCCInit = 76, CrossControl::errCodeDriverInit = 77, CrossControl-
::errCodeSetSUPPLYRESET = 78, CrossControl::errCodeRelSUPPLYRESE-
T = 79, CrossControl::errCodeSetSYSRESET = 80, CrossControl::errCode-
RelSYSRESET = 81, CrossControl::errCodeSetPWRBTN = 82, CrossControl-
::errCodeRelPWRBTN = 83, CrossControl::errCodeOnBL = 84, CrossControl-
::errCodeOffBL = 85, CrossControl::errCodeEXTFANOn = 86, CrossControl-
::errCodeEXTFANOff = 87, CrossControl::errCodePWRENOn = 88, Cross-
Control::errCodePWRENOff = 89, CrossControl::errCodeMPPWRENOn = 90,
CrossControl::errCodeMPPWRENOff = 91, CrossControl::errCodeCANPWR-
ENOn = 92, CrossControl::errCodeCANPWRENOff = 93, CrossControl::err-
CodeVID1PWRENOn = 94, CrossControl::errCodeVID1PWRENOff = 95, ×
CrossControl::errCodeVID2PWRENOn = 96, CrossControl::errCodeVID2PW-
RENOff = 97, CrossControl::errCodeVID3PWRENOn = 98, CrossControl::err-

CodeVID3PWRENOff = 99, CrossControl::errCodeVID4PWRENO = 100, -
 CrossControl::errCodeVID4PWRENOOff = 101, CrossControl::errCodeHEATA-
 CTON = 102, CrossControl::errCodeHEATACTOff = 103, CrossControl::err-
 CodeSetLEDCol = 104, CrossControl::errCodeSetLEDFreq = 105, CrossControl-
 ::errCodeManageLED = 106, CrossControl::errCodeManageCANPwr = 107, -
 CrossControl::errCodeManageMPPwr = 108, CrossControl::errCodeManageVid-
 Pwr = 109, CrossControl::errCodeManagePowSup = 110, CrossControl::errCode-
 ManageReset = 111, CrossControl::errCodeSSState = 112, CrossControl::err-
 CodeVarWrapAround = 113, CrossControl::errCodeFPBTNUnexpVal = 114, -
 CrossControl::errCodeEXTCTRLUnexpVal = 115, CrossControl::errCodeM-
 AINPWROKReadErr = 116, CrossControl::errCodeFRONTSPAREReadErr =
 117, CrossControl::errCodeTIMERReadErr = 118, CrossControl::errCodeManage-
 Diagnostics = 119, CrossControl::errCodeFPBTNTimOutReadErrEEPROM =
 120, CrossControl::errCodeEXTCTRLTimOutReadErrEEPROM = 121, Cross-
 Control::errCodeFPBTNAndExtCtrlDisabled = 122, CrossControl::errCodeSW-
 VerReadErr = 123, CrossControl::errCodeSWVerWriteErr = 124, CrossControl-
 ::errCodeManageActDeAct = 125, CrossControl::errCodeTickTimeOutTimer =
 126, CrossControl::errCodeOperateModeStateError = 127, CrossControl::err-
 CodeHeatingTempReadErrEEPROM = 128, CrossControl::errCodeMPFailedStart
 = 129, CrossControl::errCodeReadErrEEPROM = 130, CrossControl::errCode-
 TimeOutWaitingForVoltages = 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_C-
ONV [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_C-
ONV [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

ABOUTHANDLE 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)

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)

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)

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)`

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)

6.20 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)

6.20.1 Typedef Documentation

6.20.1.1 typedef `CrossControl::Video*` VIDEOHANDLE

6.20.2 Function Documentation

6.20.2.1 EXTERN_C CCAUXDLL_API VIDEOHANDLE CCAUXDLL_CALLING_CONV
GetVideo (void)

Index

- ActionShutDown
 - CrossControl, 15
- ActionSuspend
 - CrossControl, 15
- Analog_Channel_1
 - CrossControl, 17
- Analog_Channel_2
 - CrossControl, 17
- Analog_Channel_3
 - CrossControl, 17
- Analog_Channel_4
 - CrossControl, 17
- BLUE
 - CrossControl, 9
- 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, 16
- CYAN
 - CrossControl, 9
- CrossControl
 - ActionShutDown, 15
 - ActionSuspend, 15
 - Analog_Channel_1, 17
 - Analog_Channel_2, 17
 - Analog_Channel_3, 17
 - Analog_Channel_4, 17
 - BLUE, 9
 - 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, 16
 - CYAN, 9
 - DeInterlace_BOB, 9
 - DeInterlace_Even, 9
 - DeInterlace_Odd, 9
 - Disabled, 9
 - ERR_AVERAGE_CALC_STARTED, 10
 - ERR_BUFFER_SIZE, 9
 - ERR_CHECKSUM, 10
 - ERR_CODE_NOT_EXIST, 9
 - ERR_COMMAND_FAILED, 10
 - ERR_CREATE_THREAD, 10
 - ERR_DATATYPE_MISMATCH, 9
 - ERR_DEVICE_READ_DATA_FAILED, 10
 - ERR_DEVICE_WRITE_DATA_FAILED, 10
 - ERR_EEPROM, 10
 - ERR_I2C_EXPANDER_INIT_FAILED, 10
 - ERR_I2C_EXPANDER_READ_FAILED, 10
 - ERR_I2C_EXPANDER_WRITE_FAILED, 10
 - ERR_INIT_FAILED, 10
 - ERR_INVALID_DATA, 10
 - ERR_INVALID_PARAMETER, 10
 - ERR_IN_PROGRESS, 10
 - ERR_IOCTL_FAILED, 10
 - ERR_JIDA_TEMP, 10
 - ERR_MEM_ALLOC_FAIL, 10
 - ERR_NEWER_FPGA_VERSION_REQUIRED, 10

- ERR_NEWER_FRONT_VERSION_REQUIRED, 10
- ERR_NEWER_SS_VERSION_REQUIRED, 10
- ERR_NOT_RUNNING, 10
- ERR_NOT_SUPPORTED, 9
- ERR_OPEN_FAILED, 9
- ERR_SUCCESS, 9
- ERR_TELEMATICS_BT_NOT_AVAILABLE, 10
- ERR_TELEMATICS_GPRS_NOT_AVAILABLE, 10
- ERR_TELEMATICS_GPS_NOT_AVAILABLE, 10
- ERR_TELEMATICS_WLAN_NOT_AVAILABLE, 10
- ERR_UNKNOWN_FEATURE, 9
- ERR_VERIFY_FAILED, 10
- Enabled, 9
- FrameExtended, 8
- FrameStandard, 8
- FrameStandardExtended, 8
- Front_Button_Enabled, 16
- GREEN, 9
- MAGENTA, 9
- MOUSE_NEXT_BOOT, 16
- MOUSE_NOW, 16
- NoAction, 15
- OnOff_Signal_Enabled, 16
- RED, 9
- RangeExtended, 14
- RangeStandard, 14
- STD_B_D_G_H_I_N_PAL, 18
- STD_M_J_NTSC, 18
- STD_M_PAL, 18
- STD_NTSC, 18
- STD_PAL, 18
- STD_SECAM, 18
- SamplingModeAuto, 15
- SamplingModeExtended, 15
- SamplingModeStandard, 15
- TEMP_BACKPLANE, 14
- TEMP_BOARD, 14
- TEMP_BOX, 14
- TEMP_CHIPSETS, 14
- TEMP_CPU, 14
- TEMP_ENV, 14
- TEMP_OTHER, 14
- TEMP_VIDEO, 14
- TOUCH_NEXT_BOOT, 16
- TOUCH_NOW, 16
- TS_CALIBRATION_MEASUREMENTS, 17
- TS_CALIBTATION_WIDTH, 17
- TS_DEBOUNCE_TIME, 16
- TS_DEBOUNCE_TIMEOUT_TIME, 16
- TS_DOUBLECLICK_MAX_CLICK_TIME, 16
- TS_DOUBLE_CLICK_TIME, 16
- TS_LOW_LEVEL, 16
- TS_MAX_RIGHTCLICK_DISTANCE, 17
- TS_RESTORE_DEFAULT_SETTINGS, 17
- TS_RIGHT_CLICK_TIME, 16
- TS_UNTOUCHLEVEL, 16
- TS_USE_DEJITTER, 17
- UNDEFINED_COLOR, 9
- UPGRADE_COMPLETE, 17
- UPGRADE_COMPLETE_WITH_ERRORS, 17
- UPGRADE_CONVERTING_FILE, 17
- UPGRADE_FLASHING, 17
- UPGRADE_INIT, 17
- UPGRADE_PREP_COM, 17
- UPGRADE_READING_FILE, 17
- UPGRADE_VERIFYING, 17
- VOLTAGE_0V9, 18
- VOLTAGE_12V, 18
- VOLTAGE_12VID, 18
- VOLTAGE_1V0, 18
- VOLTAGE_1V05, 18
- VOLTAGE_1V2, 18
- VOLTAGE_1V5, 18
- VOLTAGE_1V8, 18
- VOLTAGE_1V9, 18
- VOLTAGE_24V, 18
- VOLTAGE_24VIN, 18
- VOLTAGE_3V3, 18
- VOLTAGE_5V, 18
- VOLTAGE_5VSTB, 18
- VOLTAGE_VREF_INT, 18
- VOLTAGE_VTFT, 18
- YELLOW, 9
- errCode0V9OutOfLimits, 12
- errCode0V9ReadErr, 12
- errCode12VOutOfLimits, 11
- errCode12VReadErr, 11
- errCode12VVIDEOOutOfLimits, 11

- errCode12VVIDEORReadErr, 11
- errCode1V05OutOfLimits, 11
- errCode1V05ReadErr, 11
- errCode1V0OutOfLimits, 12
- errCode1V0ReadErr, 11
- errCode1V2OutOfLimits, 11
- errCode1V2ReadErr, 11
- errCode1V5OutOfLimits, 11
- errCode1V5ReadErr, 11
- errCode1V8OutOfLimits, 11
- errCode1V8ReadErr, 11
- errCode1V9OutOfLimits, 11
- errCode1V9ReadErr, 11
- errCode24VINOutOfLimits, 11
- errCode24VINReadErr, 11
- errCode24VOutOfLimits, 11
- errCode24VReadErr, 11
- errCode3V3OutOfLimits, 11
- errCode3V3ReadErr, 11
- errCode5VOutOfLimits, 11
- errCode5VReadErr, 11
- errCode5VSTBYOutOfLimits, 11
- errCode5VSTBYReadErr, 11
- errCodeBLTYPEReadErrEEPROM, 12
- errCodeBLTYPEReadErrEEPROMVal, 12
- errCodeCANPWRENOFF, 13
- errCodeCANPWRENON, 13
- errCodeCANPWROKReadErr, 11
- errCodeCANPWRReadErrEEPROM, 12
- errCodeCANPWRUnexpEEPROMVal, 12
- errCodeCBRESETReadErr, 10
- errCodeDriverInit, 12
- errCodeEXTCTRLONReadErr, 11
- errCodeEXTCTRLReadErrEEPROM, 12
- errCodeEXTCTRLTimOutReadErrEEPROM, 14
- errCodeEXTCTRLUnexpEEPROMVal, 12
- errCodeEXTCTRLUnexpVal, 13
- errCodeEXTFANOFF, 13
- errCodeEXTFANON, 13
- errCodeEXTFANReadErrEEPROM, 12
- errCodeEXTFANUnexpEEPROMVal, 12
- errCodeFPBTNAndExtCtrlDisabled, 14
- errCodeFPBTNONReadErr, 11
- errCodeFPBTNReadErrEEPROM, 12
- errCodeFPBTNTimOutReadErrEEPROM, 14
- errCodeFPBTNUnexpEEPROMVal, 12
- errCodeFPBTNUnexpVal, 13
- errCodeFPGACONFReadErr, 10
- errCodeFPGACONFUnexpVal, 10
- errCodeFRONTSPAREReadErr, 13
- errCodeHEATACTOFF, 13
- errCodeHEATACTON, 13
- errCodeHeatingTempReadErrEEPROM, 14
- errCodeI2CTEMPOOutOfLimits, 12
- errCodeI2CTEMPReadErr, 12
- errCodeLEDReadErrEEPROM, 12
- errCodeLEDUnexpEEPROMVal, 12
- errCodeLVDSBLENReadErr, 11
- errCodeLVDSVDDENReadErr, 11
- errCodeLowRange24VReadErrEEPROM, 12
- errCodeLowRange24VUnexpEEPROMVal, 12
- errCodeMAINPWROKReadErr, 13
- errCodeMAX, 14
- errCodeMPFailedStart, 14
- errCodeMPPWRENOFF, 13
- errCodeMPPWRENON, 13
- errCodeManageActDeAct, 14
- errCodeManageCANPwr, 13
- errCodeManageDiagnostics, 14
- errCodeManageLED, 13
- errCodeManageMPPwr, 13
- errCodeManagePowSup, 13
- errCodeManageReset, 13
- errCodeManageVidPwr, 13
- errCodeMaxSuspTimeReadErrEEPROM, 12
- errCodeNoErr, 10
- errCodeOffBL, 13
- errCodeOnBL, 13
- errCodeOperateModeStateError, 14
- errCodePG5VSTBYReadErr, 11
- errCodePG5VSTBYUnexpVal, 11
- errCodePWRENOFF, 13
- errCodePWRENON, 13
- errCodeRCCInit, 12
- errCodeReadErrEEPROM, 14

- errCodeRelPWRBTN, 13
- errCodeRelSUPPLYRESET, 12
- errCodeRelSYSRESET, 13
- errCodeSSState, 13
- errCodeSTM32TEMPOutOfLimits, 12
- errCodeSTM32TEMPReadErr, 12
- errCodeSUS3ReadErr, 11
- errCodeSUS4ReadErr, 11
- errCodeSUS5ReadErr, 11
- errCodeSWVerReadErr, 14
- errCodeSWVerWriteErr, 14
- errCodeSetLEDCol, 13
- errCodeSetLEDFreq, 13
- errCodeSetPWRBTN, 13
- errCodeSetSUPPLYRESET, 12
- errCodeSetSYSRESET, 13
- errCodeSuspToRAMReadErrEEPROM-
M, 12
- errCodeSuspToRAMUnexpEEPROM-
Val, 12
- errCodeTFTVOLOutOfLimits, 11
- errCodeTFTVOLReadErr, 11
- errCodeTIMERReadErr, 14
- errCodeTickTimeOutTimer, 14
- errCodeTimeOutWaitingForVoltages,
14
- errCodeUnitTypeReadErrEEPROM, 12
- errCodeUnitTypeUnexpEEPROMVal,
12
- errCodeVID1PWRENOff, 13
- errCodeVID1PWRENOn, 13
- errCodeVID1PWRReadErrEEPROM,
12
- errCodeVID1PWRUnexpEEPROMVal,
12
- errCodeVID2PWRENOff, 13
- errCodeVID2PWRENOn, 13
- errCodeVID2PWRReadErrEEPROM,
12
- errCodeVID2PWRUnexpEEPROMVal,
12
- errCodeVID3PWRENOff, 13
- errCodeVID3PWRENOn, 13
- errCodeVID3PWRReadErrEEPROM,
12
- errCodeVID3PWRUnexpEEPROMVal,
12
- errCodeVID4PWRENOff, 13
- errCodeVID4PWRENOn, 13
- errCodeVID4PWRReadErrEEPROM,
12
- errCodeVID4PWRUnexpEEPROMVal,
12
- errCodeVIDPWROKReadErr, 11
- errCodeVarWrapAround, 13
- shutdownReasonCodeNoError, 15
- startupReasonCodeButtonPress, 15
- startupReasonCodeExtCtrl, 15
- startupReasonCodeMPRestart, 15
- startupReasonCodePowerOnStartup, 15
- startupReasonCodeUndefined, 15
- DeInterlace_BOB
CrossControl, 9
- DeInterlace_Even
CrossControl, 9
- DeInterlace_Odd
CrossControl, 9
- Disabled
CrossControl, 9
- ERR_AVERAGE_CALC_STARTED
CrossControl, 10
- ERR_BUFFER_SIZE
CrossControl, 9
- ERR_CHECKSUM
CrossControl, 10
- ERR_CODE_NOT_EXIST
CrossControl, 9
- ERR_COMMAND_FAILED
CrossControl, 10
- ERR_CREATE_THREAD
CrossControl, 10
- ERR_DATATYPE_MISMATCH
CrossControl, 9
- ERR_DEVICE_READ_DATA_FAILED
CrossControl, 10
- ERR_DEVICE_WRITE_DATA_FAILED
CrossControl, 10
- ERR_EEPROM
CrossControl, 10
- ERR_I2C_EXPANDER_INIT_FAILED
CrossControl, 10
- ERR_I2C_EXPANDER_READ_FAILED
CrossControl, 10
- ERR_I2C_EXPANDER_WRITE_FAILED
D
CrossControl, 10
- ERR_INIT_FAILED
CrossControl, 10
- ERR_INVALID_DATA

- CrossControl, [10](#)
- ERR_INVALID_PARAMETER
 - CrossControl, [10](#)
- ERR_IN_PROGRESS
 - CrossControl, [10](#)
- ERR_IOCTL_FAILED
 - CrossControl, [10](#)
- ERR_JIDA_TEMP
 - CrossControl, [10](#)
- ERR_MEM_ALLOC_FAIL
 - CrossControl, [10](#)
- ERR_NEWER_FPGA_VERSION_REQUIRED
 - CrossControl, [10](#)
- ERR_NEWER_FRONT_VERSION_REQUIRED
 - CrossControl, [10](#)
- ERR_NEWER_SS_VERSION_REQUIRED
 - CrossControl, [10](#)
- ERR_NOT_RUNNING
 - CrossControl, [10](#)
- ERR_NOT_SUPPORTED
 - CrossControl, [9](#)
- ERR_OPEN_FAILED
 - CrossControl, [9](#)
- ERR_SUCCESS
 - CrossControl, [9](#)
- ERR_TELEMATICS_BT_NOT_AVAILABLE
 - CrossControl, [10](#)
- ERR_TELEMATICS_GPRS_NOT_AVAILABLE
 - CrossControl, [10](#)
- ERR_TELEMATICS_GPS_NOT_AVAILABLE
 - CrossControl, [10](#)
- ERR_TELEMATICS_WLAN_NOT_AVAILABLE
 - CrossControl, [10](#)
- ERR_UNKNOWN_FEATURE
 - CrossControl, [9](#)
- ERR_VERIFY_FAILED
 - CrossControl, [10](#)
- Enabled
 - CrossControl, [9](#)
- FrameExtended
 - CrossControl, [8](#)
- FrameStandard
 - CrossControl, [8](#)
- FrameStandardExtended
 - CrossControl, [8](#)
- Front_Button_Enabled
 - CrossControl, [16](#)
- GREEN
 - CrossControl, [9](#)
- MAGENTA
 - CrossControl, [9](#)
- MOUSE_NEXT_BOOT
 - CrossControl, [16](#)
- MOUSE_NOW
 - CrossControl, [16](#)
- NoAction
 - CrossControl, [15](#)
- OnOff_Signal_Enabled
 - CrossControl, [16](#)
- RED
 - CrossControl, [9](#)
- RangeExtended
 - CrossControl, [14](#)
- RangeStandard
 - CrossControl, [14](#)
- STD_B_D_G_H_I_N_PAL
 - CrossControl, [18](#)
- STD_M_J_NTSC
 - CrossControl, [18](#)
- STD_M_PAL
 - CrossControl, [18](#)
- STD_NTSC
 - CrossControl, [18](#)
- STD_PAL
 - CrossControl, [18](#)
- STD_SECAM
 - CrossControl, [18](#)
- SamplingModeAuto
 - CrossControl, [15](#)
- SamplingModeExtended
 - CrossControl, [15](#)
- SamplingModeStandard
 - CrossControl, [15](#)
- TEMP_BACKPLANE
 - CrossControl, [14](#)
- TEMP_BOARD
 - CrossControl, [14](#)
- TEMP_BOX
 - CrossControl, [14](#)
- TEMP_CHIPSETS
 - CrossControl, [14](#)
- TEMP_CPU
 - CrossControl, [14](#)

- TEMP_ENV
 - CrossControl, [14](#)
- TEMP_OTHER
 - CrossControl, [14](#)
- TEMP_VIDEO
 - CrossControl, [14](#)
- TOUCH_NEXT_BOOT
 - CrossControl, [16](#)
- TOUCH_NOW
 - CrossControl, [16](#)
- TS_CALIBRATION_MEASUREMENTS
 - CrossControl, [17](#)
- TS_CALIBTATION_WIDTH
 - CrossControl, [17](#)
- TS_DEBOUNCE_TIME
 - CrossControl, [16](#)
- TS_DEBOUNCE_TIMEOUT_TIME
 - CrossControl, [16](#)
- TS_DOUBLECLICK_MAX_CLICK_TIME
 - CrossControl, [16](#)
- TS_DOUBLE_CLICK_TIME
 - CrossControl, [16](#)
- TS_LOW_LEVEL
 - CrossControl, [16](#)
- TS_MAX_RIGHTCLICK_DISTANCE
 - CrossControl, [17](#)
- TS_RESTORE_DEFAULT_SETTINGS
 - CrossControl, [17](#)
- TS_RIGHT_CLICK_TIME
 - CrossControl, [16](#)
- TS_UNTOUCHLEVEL
 - CrossControl, [16](#)
- TS_USE_DEJITTER
 - CrossControl, [17](#)
- UNDEFINED_COLOR
 - CrossControl, [9](#)
- UPGRADE_COMPLETE
 - CrossControl, [17](#)
- UPGRADE_COMPLETE_WITH_ERRORS
 - CrossControl, [17](#)
- UPGRADE_CONVERTING_FILE
 - CrossControl, [17](#)
- UPGRADE_FLASHING
 - CrossControl, [17](#)
- UPGRADE_INIT
 - CrossControl, [17](#)
- UPGRADE_PREP_COM
 - CrossControl, [17](#)
- CrossControl, [17](#)
- UPGRADE_READING_FILE
 - CrossControl, [17](#)
- UPGRADE_VERIFYING
 - CrossControl, [17](#)
- VOLTAGE_0V9
 - CrossControl, [18](#)
- VOLTAGE_12V
 - CrossControl, [18](#)
- VOLTAGE_12VID
 - CrossControl, [18](#)
- VOLTAGE_1V0
 - CrossControl, [18](#)
- VOLTAGE_1V05
 - CrossControl, [18](#)
- VOLTAGE_1V2
 - CrossControl, [18](#)
- VOLTAGE_1V5
 - CrossControl, [18](#)
- VOLTAGE_1V8
 - CrossControl, [18](#)
- VOLTAGE_1V9
 - CrossControl, [18](#)
- VOLTAGE_24V
 - CrossControl, [18](#)
- VOLTAGE_24VIN
 - CrossControl, [18](#)
- VOLTAGE_3V3
 - CrossControl, [18](#)
- VOLTAGE_5V
 - CrossControl, [18](#)
- VOLTAGE_5VSTB
 - CrossControl, [18](#)
- VOLTAGE_VREF_INT
 - CrossControl, [18](#)
- VOLTAGE_VTFT
 - CrossControl, [18](#)
- YELLOW
 - CrossControl, [9](#)
- ABOUTHANDLE
 - About.h, [113](#)
- ADCHANDLE
 - Adc.h, [115](#)
- AUXVERSIONHANDLE
 - AuxVersion.h, [116](#)
- About.h
 - ABOUTHANDLE, [113](#)
 - GetAbout, [113](#)
- Above80RunTime
 - CrossControl::TimerType, [95](#)

- Adc.h
 - ADCHANDLE, 115
 - GetAdc, 115
- AuxVersion.h
 - AUXVERSIONHANDLE, 116
 - GetAuxVersion, 116
- BACKLIGHTHANDLE
 - Backlight.h, 117
- BUZZERHANDLE
 - Buzzer.h, 117
- Backlight.h
 - BACKLIGHTHANDLE, 117
 - GetBacklight, 117
- ButtonPowerTransitionStatus
 - CrossControl, 8
- Buzzer.h
 - BUZZERHANDLE, 117
 - GetBuzzer, 117
- CANSETTINGHANDLE
 - CanSetting.h, 118
- CCAuxColor
 - CrossControl, 8
- CCStatus
 - CrossControl, 9
- CONFIGHANDLE
 - Config.h, 121
- CanFrameType
 - CrossControl, 8
- CanSetting.h
 - CANSETTINGHANDLE, 118
 - GetCanSetting, 118
- Config.h
 - CONFIGHANDLE, 121
 - GetConfig, 121
- CrossControl, 4
 - ButtonPowerTransitionStatus, 8
 - CCAuxColor, 8
 - CCStatus, 9
 - CanFrameType, 8
 - DeInterlaceMode, 9
 - DigitalIn_1, 20
 - DigitalIn_2, 20
 - DigitalIn_3, 20
 - DigitalIn_4, 20
 - GetErrorStringA, 18
 - GetErrorStringW, 19
 - GetHwErrorStatusStringA, 19
 - GetHwErrorStatusStringW, 19
 - GetStartupReasonStringA, 20
 - GetStartupReasonStringW, 20
 - JidaSensorType, 14
 - LightSensorOperationRange, 14
 - LightSensorSamplingMode, 14
 - PowerAction, 15
 - TSAAdvancedSettingsParameter, 16
 - TouchScreenModeSettings, 15
 - TriggerConf, 16
 - UpgradeAction, 17
 - VersionType, 8
 - Video1Conf, 20
 - Video2Conf, 20
 - Video3Conf, 21
 - Video4Conf, 21
 - VideoChannel, 17
 - VoltageEnum, 18
 - eErr, 9
 - hwErrorStatusCodes, 10
 - shutDownReasonCodes, 15
 - startupReasonCodes, 15
 - videoStandard, 17
- CrossControl::About, 22
 - Release, 33
 - getAddOnHWversion, 25
 - getAddOnManufacturingDate, 25
 - getAddOnPCBArt, 25
 - getAddOnPCBSerial, 26
 - getIsBTMounted, 26
 - getIsGPRSMounted, 27
 - getIsGPSPMounted, 27
 - getIsWLANMounted, 28
 - getMainHWversion, 28
 - getMainManufacturingDate, 28
 - getMainPCBArt, 29
 - getMainPCBSerial, 29
 - getMainProdArtNr, 30
 - getMainProdRev, 30
 - getNrOfCANConnections, 31
 - getNrOfETHConnections, 31
 - getNrOfSerialConnections, 31
 - getNrOfUSBConnections, 32
 - getNrOfVideoConnections, 32
 - getUnitSerial, 33
- CrossControl::Adc, 34
 - Release, 35
 - getVoltage, 35
- CrossControl::AuxVersion, 36
 - Release, 42
 - getCCAuxDrvVersion, 38
 - getCCAuxVersion, 39
 - getFPGAVersion, 40

- getFrontVersion, 40
- getOSVersion, 41
- getSSVersion, 42
- CrossControl::Backlight, 43
 - Release, 45
 - getAutomaticBLFilter, 43
 - getAutomaticBLParams, 44
 - getAutomaticBLStatus, 44
 - getIntensity, 44
 - getLedDimming, 45
 - getStatus, 45
 - setAutomaticBLFilter, 45
 - setAutomaticBLParams, 46
 - setIntensity, 46
 - setLedDimming, 47
 - startAutomaticBL, 47
 - stopAutomaticBL, 47
- CrossControl::Buzzer, 47
 - Release, 49
 - buzze, 48
 - getFrequency, 48
 - getTrigger, 48
 - getVolume, 49
 - setFrequency, 49
 - setTrigger, 49
 - setVolume, 50
- CrossControl::BuzzerSetup, 50
 - frequency, 50
 - volume, 50
- CrossControl::CanSetting, 51
 - Release, 52
 - getBaudrate, 51
 - getFrameType, 51
 - setBaudrate, 52
 - setFrameType, 52
- CrossControl::Config, 53
 - Release, 58
 - getCanStartupPowerConfig, 54
 - getExtFanStartupPowerConfig, 54
 - getExtOnOffSigTrigTime, 54
 - getFrontBtnTrigTime, 54
 - getHeatingTempLimit, 55
 - getLongButtonPressAction, 55
 - getOnOffSigAction, 55
 - getPowerOnStartup, 56
 - getShortButtonPressAction, 56
 - getStartupTriggerConfig, 56
 - getStartupVoltageConfig, 57
 - getSuspendMaxTime, 57
 - getVideoStartupPowerConfig, 57
 - setCanStartupPowerConfig, 58
 - setExtFanStartupPowerConfig, 58
 - setExtOnOffSigTrigTime, 59
 - setFrontBtnTrigTime, 59
 - setHeatingTempLimit, 59
 - setLongButtonPressAction, 60
 - setOnOffSigAction, 60
 - setPowerOnStartup, 60
 - setShortButtonPressAction, 61
 - setStartupTriggerConfig, 61
 - setStartupVoltageConfig, 61
 - setSuspendMaxTime, 62
 - setVideoStartupPowerConfig, 62
- CrossControl::Diagnostic, 62
 - Release, 66
 - clearHwErrorStatus, 63
 - getHwErrorStatus, 63
 - getMinMaxTemp, 63
 - getPCBTemp, 64
 - getPMTemp, 64
 - getPowerCycles, 64
 - getSSTemp, 65
 - getShutDownReason, 65
 - getStartupReason, 65
 - getTimer, 66
- CrossControl::DigIO, 66
 - Release, 68
 - getDigIO, 67
- CrossControl::FirmwareUpgrade, 68
 - Release, 69
 - getUpgradeStatus, 69
 - shutDown, 69
 - startFpgaUpgrade, 70
 - startFpgaVerification, 71
 - startFrontUpgrade, 72
 - startFrontVerification, 73
 - startSSUpgrade, 74
 - startSSVerification, 75
- CrossControl::FrontLED, 76
 - Release, 79
 - getColor, 77
 - getEnabledDuringStartup, 77
 - getIdleTime, 78
 - getNrOfPulses, 78
 - getOffTime, 78
 - getOnTime, 79
 - getSignal, 79
 - setColor, 79, 80
 - setEnabledDuringStartup, 80
 - setIdleTime, 80

- setNrOfPulses, 81
- setOff, 81
- setOffTime, 81
- setOnTime, 81
- setSignal, 82
- CrossControl::LedColorMixType, 82
 - blue, 82
 - green, 82
 - red, 82
- CrossControl::LedTimingType, 83
 - idleTime, 83
 - nrOfPulses, 83
 - offTime, 83
 - onTime, 83
- CrossControl::Lightsensor, 83
 - Release, 85
 - getAverageIlluminance, 84
 - getIlluminance, 84, 85
 - getOperatingRange, 85
 - setOperatingRange, 85
 - startAverageCalc, 86
 - stopAverageCalc, 86
- CrossControl::Power, 86
 - Release, 89
 - ackPowerRequest, 87
 - getBLPowerStatus, 87
 - getButtonPowerTransitionStatus, 87
 - getCanPowerStatus, 88
 - getExtFanPowerStatus, 88
 - getVideoPowerStatus, 88
 - setBLPowerStatus, 89
 - setCanPowerStatus, 89
 - setExtFanPowerStatus, 90
 - setVideoPowerStatus, 90
- CrossControl::Telematics, 91
 - Release, 93
 - getGPRSPowerStatus, 91
 - getGPRSStartUpPowerStatus, 92
 - getGPSAntennaStatus, 92
 - getTelematicsAvailable, 92
 - getWLANPowerStatus, 92
 - getWLANStartUpPowerStatus, 93
 - setGPRSPowerStatus, 93
 - setGPRSStartUpPowerStatus, 94
 - setWLANPowerStatus, 94
 - setWLANStartUpPowerStatus, 94
- CrossControl::TimerType, 95
 - Above80RunTime, 95
 - RunTime40_60, 95
 - RunTime60_70, 95
- RunTime70_80, 95
- TotHeatTime, 95
- TotRunTime, 95
- TotSuspTime, 96
- CrossControl::TouchScreen, 96
 - Release, 97
 - getAdvancedSetting, 96
 - getMode, 97
 - getMouseRightClickTime, 97
 - setAdvancedSetting, 97
 - setMode, 98
 - setMouseRightClickTime, 98
- CrossControl::UpgradeStatus, 98
 - currentAction, 99
 - errorCode, 99
 - percent, 99
- CrossControl::Video, 100
 - Release, 107
 - activateSnapshot, 101
 - createBitmap, 102
 - freeBmpBuffer, 102
 - getActiveChannel, 102
 - getColorKeys, 103
 - getCropping, 103
 - getDeInterlaceMode, 104
 - getDecoderReg, 103
 - getMirroring, 104
 - getRawImage, 104
 - getScaling, 105
 - getStatus, 105
 - getVideoArea, 106
 - getVideoStandard, 106
 - init, 106
 - minimize, 107
 - restore, 107
 - setActiveChannel, 107
 - setColorKeys, 108
 - setCropping, 108
 - setDeInterlaceMode, 109
 - setDecoderReg, 108
 - setMirroring, 109
 - setScaling, 109
 - setVideoArea, 110
 - showVideo, 110
 - takeSnapshot, 111
 - takeSnapshotBmp, 111
 - takeSnapshotRaw, 111
- CrossControl::received_video, 90
 - received_framerate, 91
 - received_height, 91

- received_width, 91
- CrossControl::version_info, 99
 - build, 99
 - major, 99
 - minor, 100
 - release, 100
- CrossControl::video_dec_command, 112
 - decoder_register, 112
 - register_value, 112
- DIAGNOSTICHANDLE
 - Diagnostic.h, 122
- DIGIOHANDLE
 - DigIO.h, 125
- DeInterlaceMode
 - CrossControl, 9
- Diagnostic.h
 - DIAGNOSTICHANDLE, 122
 - GetDiagnostic, 122
- DigIO.h
 - DIGIOHANDLE, 125
 - GetDigIO, 125
- DigitalIn_1
 - CrossControl, 20
- DigitalIn_2
 - CrossControl, 20
- DigitalIn_3
 - CrossControl, 20
- DigitalIn_4
 - CrossControl, 20
- FIRMWAREUPGHANDLE
 - FirmwareUpgrade.h, 126
- FRONTLEDHANDLE
 - FrontLED.h, 127
- FirmwareUpgrade.h
 - FIRMWAREUPGHANDLE, 126
 - GetFirmwareUpgrade, 126
- FrontLED.h
 - FRONTLEDHANDLE, 127
 - GetFrontLED, 127
- GetAbout
 - About.h, 113
- GetAdc
 - Adc.h, 115
- GetAuxVersion
 - AuxVersion.h, 116
- GetBacklight
 - Backlight.h, 117
- GetBuzzer
 - Buzzer.h, 117
- GetCanSetting
 - CanSetting.h, 118
- GetConfig
 - Config.h, 121
- GetDiagnostic
 - Diagnostic.h, 122
- GetDigIO
 - DigIO.h, 125
- GetErrorStringA
 - CrossControl, 18
- GetErrorStringW
 - CrossControl, 19
- GetFirmwareUpgrade
 - FirmwareUpgrade.h, 126
- GetFrontLED
 - FrontLED.h, 127
- GetHwErrorStatusStringA
 - CrossControl, 19
- GetHwErrorStatusStringW
 - CrossControl, 19
- GetLightsensor
 - Lightsensor.h, 127
- GetPower
 - Power.h, 128
- GetStartupReasonStringA
 - CrossControl, 20
- GetStartupReasonStringW
 - CrossControl, 20
- GetTelematics
 - Telematics.h, 129
- GetTouchScreen
 - TouchScreen.h, 130
- GetVideo
 - Video.h, 130
- JidaSensorType
 - CrossControl, 14
- LIGHTSENSORHANDLE
 - Lightsensor.h, 127
- LightSensorOperationRange
 - CrossControl, 14
- LightSensorSamplingMode
 - CrossControl, 14
- Lightsensor.h
 - GetLightsensor, 127
 - LIGHTSENSORHANDLE, 127
- POWERHANDLE
 - Power.h, 128
- Power.h
 - GetPower, 128
 - POWERHANDLE, 128
- PowerAction

- CrossControl, 15
- Release
 - CrossControl::About, 33
 - CrossControl::Adc, 35
 - CrossControl::AuxVersion, 42
 - CrossControl::Backlight, 45
 - CrossControl::Buzzer, 49
 - CrossControl::CanSetting, 52
 - CrossControl::Config, 58
 - CrossControl::Diagnostic, 66
 - CrossControl::DigIO, 68
 - CrossControl::FirmwareUpgrade, 69
 - CrossControl::FrontLED, 79
 - CrossControl::Lightsensor, 85
 - CrossControl::Power, 89
 - CrossControl::Telematics, 93
 - CrossControl::TouchScreen, 97
 - CrossControl::Video, 107
- RunTime40_60
 - CrossControl::TimerType, 95
- RunTime60_70
 - CrossControl::TimerType, 95
- RunTime70_80
 - CrossControl::TimerType, 95
- TELEMATICSHANDLE
 - Telematics.h, 129
- TOUCHSCREENHANDLE
 - TouchScreen.h, 130
- TSAdvancedSettingsParameter
 - CrossControl, 16
- Telematics.h
 - GetTelematics, 129
 - TELEMATICSHANDLE, 129
- TotHeatTime
 - CrossControl::TimerType, 95
- TotRunTime
 - CrossControl::TimerType, 95
- TotSuspTime
 - CrossControl::TimerType, 96
- TouchScreen.h
 - GetTouchScreen, 130
 - TOUCHSCREENHANDLE, 130
- TouchScreenModeSettings
 - CrossControl, 15
- TriggerConf
 - CrossControl, 16
- UpgradeAction
 - CrossControl, 17
- VIDEOHANDLE
 - Video.h, 130
- VersionType
 - CrossControl, 8
- Video.h
 - GetVideo, 130
 - VIDEOHANDLE, 130
- Video1Conf
 - CrossControl, 20
- Video2Conf
 - CrossControl, 20
- Video3Conf
 - CrossControl, 21
- Video4Conf
 - CrossControl, 21
- VideoChannel
 - CrossControl, 17
- VoltageEnum
 - CrossControl, 18
- ackPowerRequest
 - CrossControl::Power, 87
- activateSnapshot
 - CrossControl::Video, 101
- blue
 - CrossControl::LedColorMixType, 82
- build
 - CrossControl::version_info, 99
- buzze
 - CrossControl::Buzzer, 48
- clearHwErrorStatus
 - CrossControl::Diagnostic, 63
- createBitmap
 - CrossControl::Video, 102
- currentAction
 - CrossControl::UpgradeStatus, 99
- decoder_register
 - CrossControl::video_dec_command, 112
- eErr
 - CrossControl, 9
- errCode0V9OutOfLimits
 - CrossControl, 12
- errCode0V9ReadErr
 - CrossControl, 12
- errCode12VOutOfLimits
 - CrossControl, 11
- errCode12VReadErr
 - CrossControl, 11
- errCode12VVIDEOOutOfLimits

- CrossControl, [11](#)
- errCode12VVIDEOReadErr
 - CrossControl, [11](#)
- errCode1V05OutOfLimits
 - CrossControl, [11](#)
- errCode1V05ReadErr
 - CrossControl, [11](#)
- errCode1V00OutOfLimits
 - CrossControl, [12](#)
- errCode1V0ReadErr
 - CrossControl, [11](#)
- errCode1V2OutOfLimits
 - CrossControl, [11](#)
- errCode1V2ReadErr
 - CrossControl, [11](#)
- errCode1V5OutOfLimits
 - CrossControl, [11](#)
- errCode1V5ReadErr
 - CrossControl, [11](#)
- errCode1V8OutOfLimits
 - CrossControl, [11](#)
- errCode1V8ReadErr
 - CrossControl, [11](#)
- errCode1V9OutOfLimits
 - CrossControl, [11](#)
- errCode1V9ReadErr
 - CrossControl, [11](#)
- errCode24VINOutOfLimits
 - CrossControl, [11](#)
- errCode24VINReadErr
 - CrossControl, [11](#)
- errCode24VOutOfLimits
 - CrossControl, [11](#)
- errCode24VReadErr
 - CrossControl, [11](#)
- errCode3V3OutOfLimits
 - CrossControl, [11](#)
- errCode3V3ReadErr
 - CrossControl, [11](#)
- errCode5VOutOfLimits
 - CrossControl, [11](#)
- errCode5VReadErr
 - CrossControl, [11](#)
- errCode5VSTBYOutOfLimits
 - CrossControl, [11](#)
- errCode5VSTBYReadErr
 - CrossControl, [11](#)
- errCodeBLTYPEReadErrEEPROM
 - CrossControl, [12](#)
- errCodeBLTYPEUnexpEEPROMVal
 - CrossControl, [12](#)
- CrossControl, [12](#)
- errCodeCANPWRENOFF
 - CrossControl, [13](#)
- errCodeCANPWRENOn
 - CrossControl, [13](#)
- errCodeCANPWROKReadErr
 - CrossControl, [11](#)
- errCodeCANPWRReadErrEEPROM
 - CrossControl, [12](#)
- errCodeCANPWRUnexpEEPROMVal
 - CrossControl, [12](#)
- errCodeCBRESETReadErr
 - CrossControl, [10](#)
- errCodeDriverInit
 - CrossControl, [12](#)
- errCodeEXTCTRLONReadErr
 - CrossControl, [11](#)
- errCodeEXTCTRLReadErrEEPROM
 - CrossControl, [12](#)
- errCodeEXTCTRLTimOutReadErrEEPROM
 - CrossControl, [14](#)
- errCodeEXTCTRLUnexpEEPROMVal
 - CrossControl, [12](#)
- errCodeEXTCTRLUnexpVal
 - CrossControl, [13](#)
- errCodeEXTFANOff
 - CrossControl, [13](#)
- errCodeEXTFANOn
 - CrossControl, [13](#)
- errCodeEXTFANReadErrEEPROM
 - CrossControl, [12](#)
- errCodeEXTFANUnexpEEPROMVal
 - CrossControl, [12](#)
- errCodeFPBTNAndExtCtrlDisabled
 - CrossControl, [14](#)
- errCodeFPBTNNonReadErr
 - CrossControl, [11](#)
- errCodeFPBTNReadErrEEPROM
 - CrossControl, [12](#)
- errCodeFPBTNTimOutReadErrEEPROM
 - CrossControl, [14](#)
- errCodeFPBTNUnexpEEPROMVal
 - CrossControl, [12](#)
- errCodeFPBTNUnexpVal
 - CrossControl, [13](#)
- errCodeFPGACONFReadErr
 - CrossControl, [10](#)
- errCodeFPGACONFUnexpVal
 - CrossControl, [10](#)

- errCodeFRONTSPAREReadErr
 - CrossControl, [13](#)
- errCodeHEATACTOff
 - CrossControl, [13](#)
- errCodeHEATACTOn
 - CrossControl, [13](#)
- errCodeHeatingTempReadErrEEPROM
 - CrossControl, [14](#)
- errCodeI2CTEMPOutOfLimits
 - CrossControl, [12](#)
- errCodeI2CTEMPReadErr
 - CrossControl, [12](#)
- errCodeLEDReadErrEEPROM
 - CrossControl, [12](#)
- errCodeLEDUnexpEEPROMVal
 - CrossControl, [12](#)
- errCodeLVDSBLENReadErr
 - CrossControl, [11](#)
- errCodeLVDSVDDENReadErr
 - CrossControl, [11](#)
- errCodeLowRange24VReadErrEEPROM
 - CrossControl, [12](#)
- errCodeLowRange24VUnexpEEPROMVal
 - CrossControl, [12](#)
- errCodeMAINPWOKReadErr
 - CrossControl, [13](#)
- errCodeMAX
 - CrossControl, [14](#)
- errCodeMPFailedStart
 - CrossControl, [14](#)
- errCodeMPPWRENOff
 - CrossControl, [13](#)
- errCodeMPPWRENOn
 - CrossControl, [13](#)
- errCodeManageActDeAct
 - CrossControl, [14](#)
- errCodeManageCANPwr
 - CrossControl, [13](#)
- errCodeManageDiagnostics
 - CrossControl, [14](#)
- errCodeManageLED
 - CrossControl, [13](#)
- errCodeManageMPPwr
 - CrossControl, [13](#)
- errCodeManagePowSup
 - CrossControl, [13](#)
- errCodeManageReset
 - CrossControl, [13](#)
- errCodeManageVidPwr
 - CrossControl, [13](#)
- errCodeMaxSuspTimeReadErrEEPROM
 - CrossControl, [12](#)
- errCodeNoErr
 - CrossControl, [10](#)
- errCodeOffBL
 - CrossControl, [13](#)
- errCodeOnBL
 - CrossControl, [13](#)
- errCodeOperateModeStateError
 - CrossControl, [14](#)
- errCodePG5VSTBYReadErr
 - CrossControl, [11](#)
- errCodePG5VSTBYUnexpVal
 - CrossControl, [11](#)
- errCodePWRENOff
 - CrossControl, [13](#)
- errCodePWRENOn
 - CrossControl, [13](#)
- errCodeRCCInit
 - CrossControl, [12](#)
- errCodeReadErrEEPROM
 - CrossControl, [14](#)
- errCodeRelPWRBTN
 - CrossControl, [13](#)
- errCodeRelSUPPLYRESET
 - CrossControl, [12](#)
- errCodeRelSYSRESET
 - CrossControl, [13](#)
- errCodeSSState
 - CrossControl, [13](#)
- errCodeSTM32TEMPOutOfLimits
 - CrossControl, [12](#)
- errCodeSTM32TEMPReadErr
 - CrossControl, [12](#)
- errCodeSUS3ReadErr
 - CrossControl, [11](#)
- errCodeSUS4ReadErr
 - CrossControl, [11](#)
- errCodeSUS5ReadErr
 - CrossControl, [11](#)
- errCodeSWVerReadErr
 - CrossControl, [14](#)
- errCodeSWVerWriteErr
 - CrossControl, [14](#)
- errCodeSetLEDCol
 - CrossControl, [13](#)
- errCodeSetLEDFreq
 - CrossControl, [13](#)
- errCodeSetPWRBTN
 - CrossControl, [13](#)

- errCodeSetSUPPLYRESET
 - CrossControl, 12
- errCodeSetSYSRESET
 - CrossControl, 13
- errCodeSuspToRAMReadErrEEPROM
 - CrossControl, 12
- errCodeSuspToRAMUnexpEEPROMVal
 - CrossControl, 12
- errCodeTFTVLOOutOfLimits
 - CrossControl, 11
- errCodeTFTVOLReadErr
 - CrossControl, 11
- errCodeTIMERReadErr
 - CrossControl, 14
- errCodeTickTimeOutTimer
 - CrossControl, 14
- errCodeTimeOutWaitingForVoltages
 - CrossControl, 14
- errCodeUnitTypeReadErrEEPROM
 - CrossControl, 12
- errCodeUnitTypeUnexpEEPROMVal
 - CrossControl, 12
- errCodeVID1PWRENOff
 - CrossControl, 13
- errCodeVID1PWRENOn
 - CrossControl, 13
- errCodeVID1PWRReadErrEEPROM
 - CrossControl, 12
- errCodeVID1PWRUnexpEEPROMVal
 - CrossControl, 12
- errCodeVID2PWRENOff
 - CrossControl, 13
- errCodeVID2PWRENOn
 - CrossControl, 13
- errCodeVID2PWRReadErrEEPROM
 - CrossControl, 12
- errCodeVID2PWRUnexpEEPROMVal
 - CrossControl, 12
- errCodeVID3PWRENOff
 - CrossControl, 13
- errCodeVID3PWRENOn
 - CrossControl, 13
- errCodeVID3PWRReadErrEEPROM
 - CrossControl, 12
- errCodeVID3PWRUnexpEEPROMVal
 - CrossControl, 12
- errCodeVID4PWRENOff
 - CrossControl, 13
- errCodeVID4PWRENOn
 - CrossControl, 13
- errCodeVID4PWRReadErrEEPROM
 - CrossControl, 12
- errCodeVID4PWRUnexpEEPROMVal
 - CrossControl, 12
- errCodeVIDPWROKReadErr
 - CrossControl, 11
- errCodeVarWrapAround
 - CrossControl, 13
- errorCode
 - CrossControl::UpgradeStatus, 99
- fixedIncludeFiles/About.h, 113
- fixedIncludeFiles/Adc.h, 114
- fixedIncludeFiles/AuxVersion.h, 115
- fixedIncludeFiles/Backlight.h, 116
- fixedIncludeFiles/Buzzer.h, 117
- fixedIncludeFiles/CCAuxErrors.h, 118
- fixedIncludeFiles/CCAuxTypes.h, 119
- fixedIncludeFiles/CCPlatform.h, 120
- fixedIncludeFiles/CanSetting.h, 118
- fixedIncludeFiles/Config.h, 121
- fixedIncludeFiles/Diagnostic.h, 121
- fixedIncludeFiles/DiagnosticCodes.h, 122
- fixedIncludeFiles/DigIO.h, 124
- fixedIncludeFiles/FirmwareUpgrade.h, 125
- fixedIncludeFiles/FrontLED.h, 126
- fixedIncludeFiles/Lightsensor.h, 127
- fixedIncludeFiles/Power.h, 127
- fixedIncludeFiles/Telematics.h, 128
- fixedIncludeFiles/TouchScreen.h, 129
- fixedIncludeFiles/Video.h, 130
- freeBmpBuffer
 - CrossControl::Video, 102
- frequency
 - CrossControl::BuzzerSetup, 50
- getActiveChannel
 - CrossControl::Video, 102
- getAddOnHWversion
 - CrossControl::About, 25
- getAddOnManufacturingDate
 - CrossControl::About, 25
- getAddOnPCBArt
 - CrossControl::About, 25
- getAddOnPCBSerial
 - CrossControl::About, 26
- getAdvancedSetting
 - CrossControl::TouchScreen, 96
- getAutomaticBLFilter
 - CrossControl::Backlight, 43

- getAutomaticBLParams
 - CrossControl::Backlight, [44](#)
- getAutomaticBLStatus
 - CrossControl::Backlight, [44](#)
- getAverageIlluminance
 - CrossControl::Lightsensor, [84](#)
- getBLPowerStatus
 - CrossControl::Power, [87](#)
- getBaudrate
 - CrossControl::CanSetting, [51](#)
- getButtonPowerTransitionStatus
 - CrossControl::Power, [87](#)
- getCCAuxDrvVersion
 - CrossControl::AuxVersion, [38](#)
- getCCAuxVersion
 - CrossControl::AuxVersion, [39](#)
- getCanPowerStatus
 - CrossControl::Power, [88](#)
- getCanStartupPowerConfig
 - CrossControl::Config, [54](#)
- getColor
 - CrossControl::FrontLED, [77](#)
- getColorKeys
 - CrossControl::Video, [103](#)
- getCropping
 - CrossControl::Video, [103](#)
- getDeInterlaceMode
 - CrossControl::Video, [104](#)
- getDecoderReg
 - CrossControl::Video, [103](#)
- getDigIO
 - CrossControl::DigIO, [67](#)
- getEnabledDuringStartup
 - CrossControl::FrontLED, [77](#)
- getExtFanPowerStatus
 - CrossControl::Power, [88](#)
- getExtFanStartupPowerConfig
 - CrossControl::Config, [54](#)
- getExtOnOffSigTrigTime
 - CrossControl::Config, [54](#)
- getFPGAVersion
 - CrossControl::AuxVersion, [40](#)
- getFrameType
 - CrossControl::CanSetting, [51](#)
- getFrequency
 - CrossControl::Buzzer, [48](#)
- getFrontBtnTrigTime
 - CrossControl::Config, [54](#)
- getFrontVersion
 - CrossControl::AuxVersion, [40](#)
- getGPRSPowerStatus
 - CrossControl::Telematics, [91](#)
- getGPRSStartupPowerStatus
 - CrossControl::Telematics, [92](#)
- getGPSAntennaStatus
 - CrossControl::Telematics, [92](#)
- getHeatingTempLimit
 - CrossControl::Config, [55](#)
- getHwErrorStatus
 - CrossControl::Diagnostic, [63](#)
- getIdleTime
 - CrossControl::FrontLED, [78](#)
- getIlluminance
 - CrossControl::Lightsensor, [84](#), [85](#)
- getIntensity
 - CrossControl::Backlight, [44](#)
- getIsBTMounted
 - CrossControl::About, [26](#)
- getIsGPRSMounted
 - CrossControl::About, [27](#)
- getIsGPSMounted
 - CrossControl::About, [27](#)
- getIsWLANMounted
 - CrossControl::About, [28](#)
- getLedDimming
 - CrossControl::Backlight, [45](#)
- getLongButtonPressAction
 - CrossControl::Config, [55](#)
- getMainHWversion
 - CrossControl::About, [28](#)
- getMainManufacturingDate
 - CrossControl::About, [28](#)
- getMainPCBArt
 - CrossControl::About, [29](#)
- getMainPCBSerial
 - CrossControl::About, [29](#)
- getMainProdArtNr
 - CrossControl::About, [30](#)
- getMainProdRev
 - CrossControl::About, [30](#)
- getMinMaxTemp
 - CrossControl::Diagnostic, [63](#)
- getMirroring
 - CrossControl::Video, [104](#)
- getMode
 - CrossControl::TouchScreen, [97](#)
- getMouseRightClickTime
 - CrossControl::TouchScreen, [97](#)
- getNrOfCANConnections
 - CrossControl::About, [31](#)

- getNrOfETHConnections
 - CrossControl::About, 31
- getNrOfPulses
 - CrossControl::FrontLED, 78
- getNrOfSerialConnections
 - CrossControl::About, 31
- getNrOfUSBConnections
 - CrossControl::About, 32
- getNrOfVideoConnections
 - CrossControl::About, 32
- getOSVersion
 - CrossControl::AuxVersion, 41
- getOffTime
 - CrossControl::FrontLED, 78
- getOnOffSigAction
 - CrossControl::Config, 55
- getOnTime
 - CrossControl::FrontLED, 79
- getOperatingRange
 - CrossControl::Lightsensor, 85
- getPCBTemp
 - CrossControl::Diagnostic, 64
- getPMTemp
 - CrossControl::Diagnostic, 64
- getPowerCycles
 - CrossControl::Diagnostic, 64
- getPowerOnStartup
 - CrossControl::Config, 56
- getRawImage
 - CrossControl::Video, 104
- getSSTemp
 - CrossControl::Diagnostic, 65
- getSSVersion
 - CrossControl::AuxVersion, 42
- getScaling
 - CrossControl::Video, 105
- getShortButtonPressAction
 - CrossControl::Config, 56
- getShutDownReason
 - CrossControl::Diagnostic, 65
- getSignal
 - CrossControl::FrontLED, 79
- getStartupReason
 - CrossControl::Diagnostic, 65
- getStartupTriggerConfig
 - CrossControl::Config, 56
- getStartupVoltageConfig
 - CrossControl::Config, 57
- getStatus
 - CrossControl::Backlight, 45
- CrossControl::Video, 105
- getSuspendMaxTime
 - CrossControl::Config, 57
- getTelematicsAvailable
 - CrossControl::Telematics, 92
- getTimer
 - CrossControl::Diagnostic, 66
- getTrigger
 - CrossControl::Buzzer, 48
- getUnitSerial
 - CrossControl::About, 33
- getUpgradeStatus
 - CrossControl::FirmwareUpgrade, 69
- getVideoArea
 - CrossControl::Video, 106
- getVideoPowerStatus
 - CrossControl::Power, 88
- getVideoStandard
 - CrossControl::Video, 106
- getVideoStartupPowerConfig
 - CrossControl::Config, 57
- getVoltage
 - CrossControl::Adc, 35
- getVolume
 - CrossControl::Buzzer, 49
- getWLANPowerStatus
 - CrossControl::Telematics, 92
- getWLANStartUpPowerStatus
 - CrossControl::Telematics, 93
- green
 - CrossControl::LedColorMixType, 82
- hwErrorStatusCodes
 - CrossControl, 10
- idleTime
 - CrossControl::LedTimingType, 83
- init
 - CrossControl::Video, 106
- major
 - CrossControl::version_info, 99
- minimize
 - CrossControl::Video, 107
- minor
 - CrossControl::version_info, 100
- nrOfPulses
 - CrossControl::LedTimingType, 83
- offTime

- CrossControl::LedTimingType, 83
- onTime
 - CrossControl::LedTimingType, 83
- percent
 - CrossControl::UpgradeStatus, 99
- received_framerate
 - CrossControl::received_video, 91
- received_height
 - CrossControl::received_video, 91
- received_width
 - CrossControl::received_video, 91
- red
 - CrossControl::LedColorMixType, 82
- register_value
 - CrossControl::video_dec_command, 112
- release
 - CrossControl::version_info, 100
- restore
 - CrossControl::Video, 107
- setActiveChannel
 - CrossControl::Video, 107
- setAdvancedSetting
 - CrossControl::TouchScreen, 97
- setAutomaticBLFilter
 - CrossControl::Backlight, 45
- setAutomaticBLParams
 - CrossControl::Backlight, 46
- setBLPowerStatus
 - CrossControl::Power, 89
- setBaudrate
 - CrossControl::CanSetting, 52
- setCanPowerStatus
 - CrossControl::Power, 89
- setCanStartupPowerConfig
 - CrossControl::Config, 58
- setColor
 - CrossControl::FrontLED, 79, 80
- setColorKeys
 - CrossControl::Video, 108
- setCropping
 - CrossControl::Video, 108
- setDeInterlaceMode
 - CrossControl::Video, 109
- setDecoderReg
 - CrossControl::Video, 108
- setEnabledDuringStartup
 - CrossControl::FrontLED, 80
- setExtFanPowerStatus
 - CrossControl::Power, 90
- setExtFanStartupPowerConfig
 - CrossControl::Config, 58
- setExtOnOffSigTrigTime
 - CrossControl::Config, 59
- setFrameType
 - CrossControl::CanSetting, 52
- setFrequency
 - CrossControl::Buzzer, 49
- setFrontBtnTrigTime
 - CrossControl::Config, 59
- setGPRSPowerStatus
 - CrossControl::Telematics, 93
- setGPRSStartUpPowerStatus
 - CrossControl::Telematics, 94
- setHeatingTempLimit
 - CrossControl::Config, 59
- setIdleTime
 - CrossControl::FrontLED, 80
- setIntensity
 - CrossControl::Backlight, 46
- setLedDimming
 - CrossControl::Backlight, 47
- setLongButtonPressAction
 - CrossControl::Config, 60
- setMirroring
 - CrossControl::Video, 109
- setMode
 - CrossControl::TouchScreen, 98
- setMouseRightClickTime
 - CrossControl::TouchScreen, 98
- setNrOfPulses
 - CrossControl::FrontLED, 81
- setOff
 - CrossControl::FrontLED, 81
- setOffTime
 - CrossControl::FrontLED, 81
- setOnOffSigAction
 - CrossControl::Config, 60
- setOnTime
 - CrossControl::FrontLED, 81
- setOperatingRange
 - CrossControl::Lightsensor, 85
- setPowerOnStartup
 - CrossControl::Config, 60
- setScaling
 - CrossControl::Video, 109
- setShortButtonPressAction
 - CrossControl::Config, 61

- setSignal
 - CrossControl::FrontLED, 82
- setStartupTriggerConfig
 - CrossControl::Config, 61
- setStartupVoltageConfig
 - CrossControl::Config, 61
- setSuspendMaxTime
 - CrossControl::Config, 62
- setTrigger
 - CrossControl::Buzzer, 49
- setVideoArea
 - CrossControl::Video, 110
- setVideoPowerStatus
 - CrossControl::Power, 90
- setVideoStartupPowerConfig
 - CrossControl::Config, 62
- setVolume
 - CrossControl::Buzzer, 50
- setWLANPowerStatus
 - CrossControl::Telematics, 94
- setWLANStartUpPowerStatus
 - CrossControl::Telematics, 94
- showVideo
 - CrossControl::Video, 110
- shutDown
 - CrossControl::FirmwareUpgrade, 69
- shutDownReasonCodes
 - CrossControl, 15
- shutdownReasonCodeNoError
 - CrossControl, 15
- startAutomaticBL
 - CrossControl::Backlight, 47
- startAverageCalc
 - CrossControl::Lightsensor, 86
- startFpgaUpgrade
 - CrossControl::FirmwareUpgrade, 70
- startFpgaVerification
 - CrossControl::FirmwareUpgrade, 71
- startFrontUpgrade
 - CrossControl::FirmwareUpgrade, 72
- startFrontVerification
 - CrossControl::FirmwareUpgrade, 73
- startSSUpgrade
 - CrossControl::FirmwareUpgrade, 74
- startSSVerification
 - CrossControl::FirmwareUpgrade, 75
- startupReasonCodeButtonPress
 - CrossControl, 15
- startupReasonCodeExtCtrl
 - CrossControl, 15
- startupReasonCodeMPRestart
 - CrossControl, 15
- startupReasonCodePowerOnStartup
 - CrossControl, 15
- startupReasonCodeUndefined
 - CrossControl, 15
- startupReasonCodes
 - CrossControl, 15
- stopAutomaticBL
 - CrossControl::Backlight, 47
- stopAverageCalc
 - CrossControl::Lightsensor, 86
- takeSnapshot
 - CrossControl::Video, 111
- takeSnapshotBmp
 - CrossControl::Video, 111
- takeSnapshotRaw
 - CrossControl::Video, 111
- videoStandard
 - CrossControl, 17
- volume
 - CrossControl::BuzzerSetup, 50