

cr•ssc•ntr•l

CCAux
1.6.4.0

Tue Apr 16 2013

Contents

1	Main Page	1
1.1	Introduction	1
2	Namespace Index	2
2.1	Namespace List	2
3	Data Structure Index	3
3.1	Data Structures	3
4	File Index	4
4.1	File List	4
5	Namespace Documentation	5
5.1	CrossControl Namespace Reference	5
5.1.1	Typedef Documentation	10
5.1.1.1	_PowerMgrConf	10
5.1.1.2	_PowerMgrStatus	10
5.1.1.3	VersionType	10
5.1.2	Enumeration Type Documentation	10
5.1.2.1	ButtonPowerTransitionStatus	10
5.1.2.2	CalibrationConfigParam	11
5.1.2.3	CalibrationModeSettings	11
5.1.2.4	CanFrameType	12
5.1.2.5	CCAuxColor	12
5.1.2.6	CCStatus	12
5.1.2.7	ChargingStatus	12
5.1.2.8	DeInterlaceMode	13
5.1.2.9	eErr	13
5.1.2.10	ErrorStatus	14
5.1.2.11	hwErrorStatusCodes	14
5.1.2.12	JidaSensorType	18
5.1.2.13	LightSensorOperationRange	18
5.1.2.14	LightSensorSamplingMode	18
5.1.2.15	PowerAction	19
5.1.2.16	PowerMgrConf	19
5.1.2.17	PowerMgrStatus	19
5.1.2.18	PowerSource	19
5.1.2.19	shutDownReasonCodes	20
5.1.2.20	startupReasonCodes	20

5.1.2.21	TouchScreenModeSettings	20
5.1.2.22	TriggerConf	20
5.1.2.23	TSAdvancedSettingsParameter	21
5.1.2.24	UpgradeAction	21
5.1.2.25	VideoChannel	22
5.1.2.26	videoStandard	22
5.1.2.27	VoltageEnum	22
5.1.3	Function Documentation	23
5.1.3.1	GetErrorStringA	23
5.1.3.2	GetErrorStringW	23
5.1.3.3	GetHwErrorStatusStringA	23
5.1.3.4	GetHwErrorStatusStringW	24
5.1.3.5	GetStartupReasonStringA	24
5.1.3.6	GetStartupReasonStringW	24
5.1.4	Variable Documentation	25
5.1.4.1	DigitalIn_1	25
5.1.4.2	DigitalIn_2	25
5.1.4.3	DigitalIn_3	25
5.1.4.4	DigitalIn_4	25
5.1.4.5	Video1Conf	25
5.1.4.6	Video2Conf	25
5.1.4.7	Video3Conf	25
5.1.4.8	Video4Conf	25
6	Data Structure Documentation	26
6.1	About Struct Reference	26
6.1.1	Detailed Description	27
6.1.2	Member Function Documentation	29
6.1.2.1	getAddOnHWversion	29
6.1.2.2	getAddOnManufacturingDate	29
6.1.2.3	getAddOnPCBArt	29
6.1.2.4	getAddOnPCBSerial	30
6.1.2.5	getIsBTMounted	30
6.1.2.6	getIsDisplayAvailable	31
6.1.2.7	getIsGPRSMounted	31
6.1.2.8	getIsGPSMounted	31
6.1.2.9	getIsTouchScreenAvailable	32
6.1.2.10	getIsWLANMounted	32
6.1.2.11	getMainHWversion	33
6.1.2.12	getMainManufacturingDate	33
6.1.2.13	getMainPCBArt	33
6.1.2.14	getMainPCBSerial	34
6.1.2.15	getMainProdArtNr	34
6.1.2.16	getMainProdRev	35
6.1.2.17	getNrOfCANConnections	35
6.1.2.18	getNrOfDigIOConnections	36
6.1.2.19	getNrOfETHConnections	36
6.1.2.20	getNrOfSerialConnections	36
6.1.2.21	getNrOfUSBConnections	37
6.1.2.22	getNrOfVideoConnections	37

6.1.2.23	getUnitSerial	38
6.1.2.24	Release	38
6.2	Adc Struct Reference	38
6.2.1	Detailed Description	39
6.2.2	Member Function Documentation	40
6.2.2.1	getVoltage	40
6.2.2.2	Release	40
6.3	AuxVersion Struct Reference	41
6.3.1	Detailed Description	41
6.3.2	Member Function Documentation	43
6.3.2.1	getCCAuxDrvVersion	43
6.3.2.2	getCCAuxVersion	43
6.3.2.3	getFPGAVersion	44
6.3.2.4	getFrontVersion	45
6.3.2.5	getOSVersion	45
6.3.2.6	getSSVersion	46
6.3.2.7	Release	46
6.4	Backlight Struct Reference	47
6.4.1	Detailed Description	47
6.4.2	Member Function Documentation	48
6.4.2.1	getAutomaticBLFilter	49
6.4.2.2	getAutomaticBLParams	49
6.4.2.3	getAutomaticBLStatus	49
6.4.2.4	getIntensity	50
6.4.2.5	getLedDimming	50
6.4.2.6	getStatus	50
6.4.2.7	Release	51
6.4.2.8	setAutomaticBLFilter	51
6.4.2.9	setAutomaticBLParams	52
6.4.2.10	setIntensity	52
6.4.2.11	setLedDimming	52
6.4.2.12	startAutomaticBL	53
6.4.2.13	stopAutomaticBL	53
6.5	Battery Struct Reference	53
6.5.1	Detailed Description	54
6.5.2	Member Function Documentation	58
6.5.2.1	getBatteryChargingStatus	58
6.5.2.2	getBatteryHWversion	58
6.5.2.3	getBatterySerial	59
6.5.2.4	getBatterySwVersion	60
6.5.2.5	getBatteryTemp	60
6.5.2.6	getBatteryVoltageStatus	61
6.5.2.7	getHwErrorStatus	62
6.5.2.8	getMinMaxTemp	62
6.5.2.9	getPowerSource	63
6.5.2.10	getTimer	64
6.5.2.11	isBatteryPresent	64
6.5.2.12	Release	65
6.6	BatteryTimerType Struct Reference	65
6.6.1	Detailed Description	66

6.6.2	Field Documentation	66
6.6.2.1	RunTime_0_40	66
6.6.2.2	RunTime_40_60	66
6.6.2.3	RunTime_60_70	66
6.6.2.4	RunTime_70_80	66
6.6.2.5	RunTime_Above80	66
6.6.2.6	RunTime_m20	66
6.6.2.7	RunTime_m20_0	66
6.6.2.8	TotRunTimeBattery	66
6.6.2.9	TotRunTimeMain	66
6.7	Buzzer Struct Reference	67
6.7.1	Detailed Description	67
6.7.2	Member Function Documentation	68
6.7.2.1	buzze	68
6.7.2.2	getFrequency	69
6.7.2.3	getTrigger	69
6.7.2.4	getVolume	69
6.7.2.5	Release	70
6.7.2.6	setFrequency	70
6.7.2.7	setTrigger	71
6.7.2.8	setVolume	71
6.8	BuzzerSetup Struct Reference	71
6.8.1	Field Documentation	72
6.8.1.1	frequency	72
6.8.1.2	volume	72
6.9	CanSetting Struct Reference	72
6.9.1	Detailed Description	72
6.9.2	Member Function Documentation	73
6.9.2.1	getBaudrate	73
6.9.2.2	getFrameType	74
6.9.2.3	Release	74
6.9.2.4	setBaudrate	75
6.9.2.5	setFrameType	75
6.10	Config Struct Reference	75
6.10.1	Detailed Description	76
6.10.2	Member Function Documentation	77
6.10.2.1	getCanStartupPowerConfig	77
6.10.2.2	getExtFanStartupPowerConfig	78
6.10.2.3	getExtOnOffSigTrigTime	78
6.10.2.4	getFrontBtnTrigTime	78
6.10.2.5	getHeatingTempLimit	79
6.10.2.6	getLongButtonPressAction	79
6.10.2.7	getOnOffSigAction	79
6.10.2.8	getPowerOnStartup	80
6.10.2.9	getShortButtonPressAction	80
6.10.2.10	getStartupTriggerConfig	80
6.10.2.11	getStartupVoltageConfig	81
6.10.2.12	getSuspendMaxTime	81
6.10.2.13	getVideoStartupPowerConfig	82
6.10.2.14	Release	82

6.10.2.15	setCanStartupPowerConfig	82
6.10.2.16	setExtFanStartupPowerConfig	83
6.10.2.17	setExtOnOffSigTrigTime	83
6.10.2.18	setFrontBtnTrigTime	83
6.10.2.19	setHeatingTempLimit	84
6.10.2.20	setLongButtonPressAction	84
6.10.2.21	setOnOffSigAction	84
6.10.2.22	setPowerOnStartup	85
6.10.2.23	setShortButtonPressAction	85
6.10.2.24	setStartupTriggerConfig	86
6.10.2.25	setStartupVoltageConfig	86
6.10.2.26	setSuspendMaxTime	86
6.10.2.27	setVideoStartupPowerConfig	87
6.11	Diagnostic Struct Reference	87
6.11.1	Detailed Description	88
6.11.2	Member Function Documentation	89
6.11.2.1	clearHwErrorStatus	89
6.11.2.2	getHwErrorStatus	89
6.11.2.3	getMinMaxTemp	89
6.11.2.4	getPCBTemp	90
6.11.2.5	getPMTemp	90
6.11.2.6	getPowerCycles	90
6.11.2.7	getShutDownReason	91
6.11.2.8	getSSTemp	91
6.11.2.9	getStartupReason	91
6.11.2.10	getTimer	92
6.11.2.11	Release	92
6.12	DigIO Struct Reference	92
6.12.1	Detailed Description	93
6.12.2	Member Function Documentation	93
6.12.2.1	getDigIO	94
6.12.2.2	getDigIODir	94
6.12.2.3	Release	95
6.12.2.4	setDigIO	95
6.12.2.5	setDigIODir	95
6.13	FirmwareUpgrade Struct Reference	96
6.13.1	Detailed Description	96
6.13.2	Member Function Documentation	98
6.13.2.1	getUpgradeStatus	99
6.13.2.2	Release	99
6.13.2.3	shutDown	99
6.13.2.4	startFpgaUpgrade	99
6.13.2.5	startFpgaVerification	100
6.13.2.6	startFrontUpgrade	101
6.13.2.7	startFrontVerification	102
6.13.2.8	startSSUpgrade	103
6.13.2.9	startSSVerification	104
6.14	FrontLED Struct Reference	105
6.14.1	Detailed Description	106
6.14.2	Member Function Documentation	107

6.14.2.1	getColor	107
6.14.2.2	getColor	108
6.14.2.3	getEnabledDuringStartup	108
6.14.2.4	getIdleTime	108
6.14.2.5	getNrOfPulses	109
6.14.2.6	getOffTime	109
6.14.2.7	getOnTime	109
6.14.2.8	getSignal	110
6.14.2.9	Release	110
6.14.2.10	setColor	110
6.14.2.11	setColor	111
6.14.2.12	setEnabledDuringStartup	111
6.14.2.13	setIdleTime	111
6.14.2.14	setNrOfPulses	112
6.14.2.15	setOff	112
6.14.2.16	setOffTime	112
6.14.2.17	setOnTime	113
6.14.2.18	setSignal	113
6.15	LedColorMixType Struct Reference	113
6.15.1	Field Documentation	114
6.15.1.1	blue	114
6.15.1.2	green	114
6.15.1.3	red	114
6.16	LedTimingType Struct Reference	114
6.16.1	Field Documentation	114
6.16.1.1	idleTime	114
6.16.1.2	nrOfPulses	114
6.16.1.3	offTime	115
6.16.1.4	onTime	115
6.17	Lightsensor Struct Reference	115
6.17.1	Detailed Description	115
6.17.2	Member Function Documentation	116
6.17.2.1	getAverageIlluminance	117
6.17.2.2	getIlluminance	117
6.17.2.3	getIlluminance	117
6.17.2.4	getOperatingRange	118
6.17.2.5	Release	118
6.17.2.6	setOperatingRange	118
6.17.2.7	startAverageCalc	119
6.17.2.8	stopAverageCalc	119
6.18	Power Struct Reference	120
6.18.1	Detailed Description	120
6.18.2	Member Function Documentation	121
6.18.2.1	ackPowerRequest	121
6.18.2.2	getBLPowerStatus	122
6.18.2.3	getButtonPowerTransitionStatus	122
6.18.2.4	getCanPowerStatus	122
6.18.2.5	getExtFanPowerStatus	123
6.18.2.6	getVideoPowerStatus	123
6.18.2.7	Release	124

6.18.2.8	setBLPowerStatus	124
6.18.2.9	setCanPowerStatus	124
6.18.2.10	setExtFanPowerStatus	125
6.18.2.11	setVideoPowerStatus	125
6.19	PowerMgr Struct Reference	125
6.19.1	Detailed Description	126
6.19.2	Member Function Documentation	129
6.19.2.1	getConfiguration	129
6.19.2.2	getPowerMgrStatus	130
6.19.2.3	hasResumed	132
6.19.2.4	registerControlledSuspendOrShutDown	134
6.19.2.5	Release	135
6.19.2.6	setAppReadyForSuspendOrShutdown	135
6.20	received_video Struct Reference	137
6.20.1	Field Documentation	138
6.20.1.1	received_framerate	138
6.20.1.2	received_height	138
6.20.1.3	received_width	138
6.21	Smart Struct Reference	138
6.21.1	Detailed Description	138
6.21.2	Member Function Documentation	139
6.21.2.1	getDeviceSerial	139
6.21.2.2	getInitialTime	140
6.21.2.3	getRemainingLifeTime	141
6.21.2.4	Release	141
6.22	Telematics Struct Reference	141
6.22.1	Detailed Description	142
6.22.2	Member Function Documentation	145
6.22.2.1	getBTPowerStatus	145
6.22.2.2	getBTStartUpPowerStatus	145
6.22.2.3	getGPRSPowerStatus	146
6.22.2.4	getGPRSStartUpPowerStatus	146
6.22.2.5	getGPSAntennaStatus	147
6.22.2.6	getGPSPowerStatus	147
6.22.2.7	getGPSStartUpPowerStatus	148
6.22.2.8	getTelematicsAvailable	149
6.22.2.9	getWLANPowerStatus	149
6.22.2.10	getWLANStartUpPowerStatus	150
6.22.2.11	Release	150
6.22.2.12	setBTPowerStatus	150
6.22.2.13	setBTStartUpPowerStatus	151
6.22.2.14	setGPRSPowerStatus	151
6.22.2.15	setGPRSStartUpPowerStatus	151
6.22.2.16	setGPSPowerStatus	152
6.22.2.17	setGPSStartUpPowerStatus	152
6.22.2.18	setWLANPowerStatus	152
6.22.2.19	setWLANStartUpPowerStatus	152
6.23	TimerType Struct Reference	153
6.23.1	Detailed Description	153
6.23.2	Field Documentation	153

6.23.2.1	Above80RunTime	153
6.23.2.2	RunTime40_60	153
6.23.2.3	RunTime60_70	153
6.23.2.4	RunTime70_80	153
6.23.2.5	TotHeatTime	154
6.23.2.6	TotRunTime	154
6.23.2.7	TotSuspTime	154
6.24	TouchScreen Struct Reference	154
6.24.1	Detailed Description	154
6.24.2	Member Function Documentation	155
6.24.2.1	getAdvancedSetting	155
6.24.2.2	getMode	156
6.24.2.3	getMouseRightClickTime	157
6.24.2.4	Release	157
6.24.2.5	setAdvancedSetting	157
6.24.2.6	setMode	158
6.24.2.7	setMouseRightClickTime	158
6.25	TouchScreenCalib Struct Reference	158
6.25.1	Detailed Description	159
6.25.2	Member Function Documentation	159
6.25.2.1	CheckCalibrationPointFinished	159
6.25.2.2	GetConfigParam	159
6.25.2.3	GetMode	160
6.25.2.4	Release	160
6.25.2.5	SetCalibrationPoint	160
6.25.2.6	SetConfigParam	160
6.25.2.7	SetMode	161
6.26	UpgradeStatus Struct Reference	161
6.26.1	Detailed Description	161
6.26.2	Field Documentation	161
6.26.2.1	currentAction	161
6.26.2.2	errorCode	161
6.26.2.3	percent	162
6.27	version_info Struct Reference	162
6.27.1	Field Documentation	162
6.27.1.1	build	162
6.27.1.2	major	162
6.27.1.3	minor	162
6.27.1.4	release	162
6.28	Video Struct Reference	162
6.28.1	Detailed Description	164
6.28.2	Member Function Documentation	164
6.28.2.1	activateSnapshot	164
6.28.2.2	createBitmap	164
6.28.2.3	freeBmpBuffer	165
6.28.2.4	getActiveChannel	165
6.28.2.5	getColorKeys	165
6.28.2.6	getCropping	165
6.28.2.7	getDecoderReg	166
6.28.2.8	getDeInterlaceMode	166

6.28.2.9	getMirroring	166
6.28.2.10	getRawImage	167
6.28.2.11	getScaling	167
6.28.2.12	getStatus	167
6.28.2.13	getVideoArea	168
6.28.2.14	getVideoStandard	168
6.28.2.15	init	169
6.28.2.16	minimize	169
6.28.2.17	Release	169
6.28.2.18	restore	169
6.28.2.19	setActiveChannel	170
6.28.2.20	setColorKeys	170
6.28.2.21	setCropping	170
6.28.2.22	setDecoderReg	171
6.28.2.23	setDeInterlaceMode	171
6.28.2.24	setMirroring	171
6.28.2.25	setScaling	172
6.28.2.26	setVideoArea	172
6.28.2.27	showVideo	172
6.28.2.28	takeSnapshot	173
6.28.2.29	takeSnapshotBmp	173
6.28.2.30	takeSnapshotRaw	174
6.29	video_dec_command Struct Reference	174
6.29.1	Field Documentation	174
6.29.1.1	decoder_register	174
6.29.1.2	register_value	174
7	File Documentation	176
7.1	IncludeFiles/About.h File Reference	176
7.1.1	Typedef Documentation	176
7.1.1.1	ABOUTHANDLE	176
7.1.2	Function Documentation	176
7.1.2.1	GetAbout	176
7.2	IncludeFiles/Adc.h File Reference	177
7.2.1	Typedef Documentation	178
7.2.1.1	ADCHANDLE	178
7.2.2	Function Documentation	178
7.2.2.1	GetAdc	178
7.3	IncludeFiles/AuxVersion.h File Reference	178
7.3.1	Typedef Documentation	179
7.3.1.1	AUXVERSIONHANDLE	179
7.3.2	Function Documentation	179
7.3.2.1	GetAuxVersion	179
7.4	IncludeFiles/Backlight.h File Reference	179
7.4.1	Typedef Documentation	180
7.4.1.1	BACKLIGHTHANDLE	180
7.4.2	Function Documentation	180
7.4.2.1	GetBacklight	180
7.5	IncludeFiles/Battery.h File Reference	180
7.5.1	Typedef Documentation	181

7.5.1.1	BATTERYHANDLE	181
7.5.2	Function Documentation	181
7.5.2.1	GetBattery	181
7.6	IncludeFiles/Buzzer.h File Reference	181
7.6.1	Typedef Documentation	182
7.6.1.1	BUZZERHANDLE	182
7.6.2	Function Documentation	182
7.6.2.1	GetBuzzer	182
7.7	IncludeFiles/CanSetting.h File Reference	182
7.7.1	Typedef Documentation	183
7.7.1.1	CANSETTINGHANDLE	183
7.7.2	Function Documentation	183
7.7.2.1	GetCanSetting	183
7.8	IncludeFiles/CCAuxErrors.h File Reference	183
7.9	IncludeFiles/CCAuxTypes.h File Reference	184
7.10	IncludeFiles/CCPlatform.h File Reference	185
7.11	IncludeFiles/Config.h File Reference	185
7.11.1	Typedef Documentation	186
7.11.1.1	CONFIGHANDLE	186
7.11.2	Function Documentation	186
7.11.2.1	GetConfig	186
7.12	IncludeFiles/Diagnostic.h File Reference	187
7.12.1	Typedef Documentation	187
7.12.1.1	DIAGNOSTICHANDLE	187
7.12.2	Function Documentation	187
7.12.2.1	GetDiagnostic	187
7.13	IncludeFiles/DiagnosticCodes.h File Reference	188
7.14	IncludeFiles/DigIO.h File Reference	190
7.14.1	Typedef Documentation	191
7.14.1.1	DIGIOHANDLE	191
7.14.2	Function Documentation	191
7.14.2.1	GetDigIO	191
7.15	IncludeFiles/FirmwareUpgrade.h File Reference	191
7.15.1	Typedef Documentation	192
7.15.1.1	FIRMWAREUPGHANDLE	192
7.15.2	Function Documentation	192
7.15.2.1	GetFirmwareUpgrade	192
7.16	IncludeFiles/FrontLED.h File Reference	192
7.16.1	Typedef Documentation	193
7.16.1.1	FRONTLEDHANDLE	193
7.16.2	Function Documentation	193
7.16.2.1	GetFrontLED	193
7.17	IncludeFiles/Lightsensor.h File Reference	193
7.17.1	Typedef Documentation	194
7.17.1.1	LIGHTSENSORHANDLE	194
7.17.2	Function Documentation	194
7.17.2.1	GetLightsensor	194
7.18	IncludeFiles/Power.h File Reference	194
7.18.1	Typedef Documentation	195
7.18.1.1	POWERHANDLE	195

7.18.2	Function Documentation	195
7.18.2.1	GetPower	195
7.19	IncludeFiles/PowerMgr.h File Reference	195
7.19.1	Typedef Documentation	196
7.19.1.1	POWERMGRHANDLE	196
7.19.2	Function Documentation	196
7.19.2.1	GetPowerMgr	196
7.20	IncludeFiles/Releasenotes.dox File Reference	197
7.21	IncludeFiles/Smart.h File Reference	197
7.21.1	Typedef Documentation	197
7.21.1.1	SMARTHANDLE	197
7.21.2	Function Documentation	197
7.21.2.1	GetSmart	197
7.22	IncludeFiles/Telematics.h File Reference	198
7.22.1	Typedef Documentation	198
7.22.1.1	TELEMATICSHANDLE	198
7.22.2	Function Documentation	198
7.22.2.1	GetTelematics	198
7.23	IncludeFiles/TouchScreen.h File Reference	199
7.23.1	Typedef Documentation	200
7.23.1.1	TOUCHSCREENHANDLE	200
7.23.2	Function Documentation	200
7.23.2.1	GetTouchScreen	200
7.24	IncludeFiles/TouchScreenCalib.h File Reference	200
7.24.1	Typedef Documentation	201
7.24.1.1	TOUCHSCREENCALIBHANDLE	201
7.24.2	Function Documentation	201
7.24.2.1	GetTouchScreenCalib	201
7.25	IncludeFiles/Video.h File Reference	201
7.25.1	Typedef Documentation	202
7.25.1.1	VIDEOHANDLE	202
7.25.2	Function Documentation	202
7.25.2.1	GetVideo	202

Chapter 1

Main Page

1.1 Introduction

This documentation is generated from the CCAux source code. CCAux ([CrossControl Common Aux control](#)) is an API that gives access to settings, features and many hardware interfaces; backlight, buzzer, diagnostics, frontled, lightsensor and analog video interfaces.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

CrossControl	5
--	---

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

About	26
Adc	38
AuxVersion	41
Backlight	47
Battery	53
BatteryTimerType	65
Buzzer	67
BuzzerSetup	71
CanSetting	72
Config	75
Diagnostic	87
DigIO	92
FirmwareUpgrade	96
FrontLED	105
LedColorMixType	113
LedTimingType	114
Lightsensor	115
Power	120
PowerMgr	125
received_video	137
Smart	138
Telematics	141
TimerType	153
TouchScreen	154
TouchScreenCalib	158
UpgradeStatus	161
version_info	162
Video	162
video_dec_command	174

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

IncludeFiles/About.h	176
IncludeFiles/Adc.h	177
IncludeFiles/AuxVersion.h	178
IncludeFiles/Backlight.h	179
IncludeFiles/Battery.h	180
IncludeFiles/Buzzer.h	181
IncludeFiles/CanSetting.h	182
IncludeFiles/CCAuxErrors.h	183
IncludeFiles/CCAuxTypes.h	184
IncludeFiles/CCPlatform.h	185
IncludeFiles/Config.h	185
IncludeFiles/Diagnostic.h	187
IncludeFiles/DiagnosticCodes.h	188
IncludeFiles/DigIO.h	190
IncludeFiles/FirmwareUpgrade.h	191
IncludeFiles/FrontLED.h	192
IncludeFiles/Lightsensor.h	193
IncludeFiles/Power.h	194
IncludeFiles/PowerMgr.h	195
IncludeFiles/Smart.h	197
IncludeFiles/Telematics.h	198
IncludeFiles/TouchScreen.h	199
IncludeFiles/TouchScreenCalib.h	200
IncludeFiles/Video.h	201

Chapter 5

Namespace Documentation

5.1 CrossControl Namespace Reference

Data Structures

- struct [About](#)
- struct [Adc](#)
- struct [AuxVersion](#)
- struct [Backlight](#)
- struct [BatteryTimerType](#)
- struct [Battery](#)
- 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 [PowerMgr](#)
- struct [Smart](#)
- struct [Telematics](#)
- struct [TouchScreen](#)
- struct [TouchScreenCalib](#)
- struct [Video](#)

Typedefs

- typedef struct `version_info` `VersionType`
- typedef enum `CrossControl::PowerMgrConf` `_PowerMgrConf`
- typedef enum `CrossControl::PowerMgrStatus` `_PowerMgrStatus`

Enumerations

- enum `VoltageEnum` {
`VOLTAGE_24VIN` = 0, `VOLTAGE_24V`, `VOLTAGE_12V`, `VOLTAGE_12-VID`,
`VOLTAGE_5V`, `VOLTAGE_3V3`, `VOLTAGE_VTFT`, `VOLTAGE_5VSTB`,
`VOLTAGE_1V9`, `VOLTAGE_1V8`, `VOLTAGE_1V5`, `VOLTAGE_1V2`,
`VOLTAGE_1V05`, `VOLTAGE_1V0`, `VOLTAGE_0V9`, `VOLTAGE_VREF_I-NT` }
- enum `ChargingStatus` {
`ChargingStatus_NoCharge` = 0, `ChargingStatus_Charging` = 1, `ChargingStatus-
_FullyCharged` = 2, `ChargingStatus_TempLow` = 3,
`ChargingStatus_TempHigh` = 4, `ChargingStatus_Unknown` = 5 }
- enum `PowerSource` { `PowerSource_Battery` = 0, `PowerSource_ExternalPower` = 1 }
- enum `ErrorStatus` {
`ErrorStatus_NoError` = 0, `ErrorStatus_ThermistorTempSensor` = 1, `ErrorStatus-
_SecondaryTempSensor` = 2, `ErrorStatus_ChargeFail` = 3,
`ErrorStatus_Overcurrent` = 4, `ErrorStatus_Init` = 5 }
- enum `LightSensorOperationRange` { `RangeStandard` = 0, `RangeExtended` = 1 }
- enum `LightSensorSamplingMode` { `SamplingModeStandard` = 0, `SamplingMode-
_Extended`, `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_CRE-
ATE_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_D_PAL = 3, STD_NTSC = 4, STD_SECAM = 5` }
- enum `CanFrameType` { `FrameStandard, FrameExtended, FrameStandardExtended` }
- enum `TriggerConf` { `Front_Button_Enabled = 1, OnOff_Signal_Enabled = 2, Both_Button_And_Signal_Enabled = 3` }
- enum `PowerAction` { `NoAction = 0, ActionSuspend = 1, ActionShutDown = 2` }
- enum `ButtonPowerTransitionStatus` { `BPTS_No_Change = 0, BPTS_ShutDown = 1, BPTS_Suspend = 2, BPTS_Restart = 3, BPTS_BtnPressed = 4, BPTS_BtnPressedLong = 5, BPTS_SignalOff = 6` }
- enum `JidaSensorType` { `TEMP_CPU = 0, TEMP_BOX = 1, TEMP_ENV = 2, TEMP_BOARD = 3, TEMP_BACKPLANE = 4, TEMP_CHIPSETS = 5, TEMP_VIDEO = 6, TEMP_OTHER = 7` }
- enum `UpgradeAction` { `UPGRADE_INIT, UPGRADE_PREP_COM, UPGRADE_READING_FILE, UPGRADE_CONVERTING_FILE, UPGRADE_FLASHING, UPGRADE_VERIFYING, UPGRADE_COMPLETE, UPGRADE_COMPLETE_WITH_ERRORS` }
- enum `CCAuxColor` { `RED = 0, GREEN, BLUE, CYAN, MAGENTA, YELLOW, UNDEFINED_COLOR` }
- enum `startupReasonCodes` { `startupReasonCodeUndefined = 0x0000, startupReasonCodeButtonPress = 0x0055, startupReasonCodeExtCtrl = 0x00AA, startupReasonCodeMPRestart = 0x00F0, startupReasonCodePowerOnStartup = 0x000F` }
- enum `shutDownReasonCodes` { `shutdownReasonCodeNoError = 0x001F` }
- enum `hwErrorStatusCodes` { `errCodeNoErr = 0, errCodeFPGACONFReadErr = 1, errCodeFPGACONFUnexpVal = 2, errCodeCBRESETReadErr = 3, errCodeSUS3ReadErr = 4, errCodeSUS4ReadErr = 5, errCodeSUS5ReadErr = 6, errCodePG5VSTBYReadErr = 7, errCodePG5VSTBYUnexpVal = 8, errCodeCANPWROKReadErr = 9, errCodeVIDPWROKReadErr = 10, errCodeLVDSBLENReadErr = 11, errCodeLVDSVD DENReadErr = 12, errCodeEXTCTRLONReadErr = 13, errCodeFPBTNONReadErr = 14, errCode24VReadErr = 15, errCode24VOutOfLimits = 16, errCode24VINReadErr = 17, errCode24VINOutOfLimits = 18, errCode12VReadErr = 19, errCode12VOutOfLimits = 20, errCode12VVIDEORReadErr = 21, errCode12V-`

VIDEOOutOfLimits = 22, errCode5VSTBYReadErr = 23,
errCode5VSTBYOutOfLimits = 24, errCode5VReadErr = 25, errCode5VOut-
OfLimits = 26, errCode3V3ReadErr = 27,
errCode3V3OutOfLimits = 28, errCodeTFTVOLReadErr = 29, errCodeTFTV-
OLOOutOfLimits = 30, errCode1V9ReadErr = 31,
errCode1V9OutOfLimits = 32, errCode1V8ReadErr = 33, errCode1V8Out-
OfLimits = 34, errCode1V5ReadErr = 35,
errCode1V5OutOfLimits = 36, errCode1V2ReadErr = 37, errCode1V2Out-
OfLimits = 38, errCode1V05ReadErr = 39,
errCode1V05OutOfLimits = 40, errCode1V0ReadErr = 41, errCode1V0Out-
OfLimits = 42, errCode0V9ReadErr = 43,
errCode0V9OutOfLimits = 44, errCodeI2CTEMPReadErr = 45, errCodeI2CT-
EMPOutOfLimits = 46, errCodeSTM32TEMPReadErr = 47,
errCodeSTM32TEMPOutOfLimits = 48, errCodeBLTYPEUnexpEEPROMVal
= 49, errCodeFPBTNUnexpEEPROMVal = 50, errCodeEXTCTRLUnexpEEP-
ROMVal = 51,
errCodeLowRange24VUnexpEEPROMVal = 52, errCodeSuspToRAMUnexpE-
EPROMVal = 53, errCodeCANPWRUnexpEEPROMVal = 54, errCodeVID1P-
WRUnexpEEPROMVal = 55,
errCodeVID2PWRUnexpEEPROMVal = 56, errCodeVID3PWRUnexpEEPRO-
MVal = 57, errCodeVID4PWRUnexpEEPROMVal = 58, errCodeEXTFANUnexp-
EEPROMVal = 59,
errCodeLEDUnexpEEPROMVal = 60, errCodeUnitTypeUnexpEEPROMVal =
61, errCodeBLTYPEReadErrEEPROM = 62, errCodeFPBTNReadErrEEPRO-
M = 63,
errCodeEXTCTRLReadErrEEPROM = 64, errCodeMaxSuspTimeReadErrEE-
EPROM = 65, errCodeLowRange24VReadErrEEPROM = 66, errCodeSuspToR-
AMReadErrEEPROM = 67,
errCodeCANPWRReadErrEEPROM = 68, errCodeVID1PWRReadErrEEPRO-
M = 69, errCodeVID2PWRReadErrEEPROM = 70, errCodeVID3PWRRead-
ErrEEPROM = 71,
errCodeVID4PWRReadErrEEPROM = 72, errCodeEXTFANReadErrEEPRO-
M = 73, errCodeLEDReadErrEEPROM = 74, errCodeUnitTypeReadErrEEPR-
OM = 75,
errCodeRCCInit = 76, errCodeDriverInit = 77, errCodeSetSUPPLYRESET =
78, errCodeRelSUPPLYRESET = 79,
errCodeSetSYSRESET = 80, errCodeRelSYSRESET = 81, errCodeSetPWRB-
TN = 82, errCodeRelPWRBTN = 83,
errCodeOnBL = 84, errCodeOffBL = 85, errCodeEXTFANOn = 86, errCodeE-
XTFANOff = 87,
errCodePWRENOn = 88, errCodePWRENOff = 89, errCodeMPPWRENOn =
90, errCodeMPPWRENOff = 91,
errCodeCANPWRENOn = 92, errCodeCANPWRENOff = 93, errCodeVID1P-
WRENOn = 94, errCodeVID1PWRENOff = 95,
errCodeVID2PWRENOn = 96, errCodeVID2PWRENOff = 97, errCodeVID3P-
WRENOn = 98, errCodeVID3PWRENOff = 99,
errCodeVID4PWRENOn = 100, errCodeVID4PWRENOff = 101, errCodeHE-
ATACTOn = 102, errCodeHEATACTOff = 103,
errCodeSetLEDCol = 104, errCodeSetLEDFreq = 105, errCodeManageLED =

- 106, `errCodeManageCANPwr` = 107,
`errCodeManageMPPwr` = 108, `errCodeManageVidPwr` = 109, `errCodeManagePowSup` = 110, `errCodeManageReset` = 111,
`errCodeSSState` = 112, `errCodeVarWrapAround` = 113, `errCodeFPBTNUexpVal` = 114, `errCodeEXTCTRLUnexpVal` = 115,
`errCodeMAINPWOKReadErr` = 116, `errCodeFRONTSPAREReadErr` = 117,
`errCodeTIMERReadErr` = 118, `errCodeManageDiagnostics` = 119,
`errCodeFPBTNTimOutReadErrEEPROM` = 120, `errCodeEXTCTRLTimOutReadErrEEPROM` = 121, `errCodeFPBTNAndExtCtrlDisabled` = 122, `errCodeSWVerReadErr` = 123,
`errCodeSWVerWriteErr` = 124, `errCodeManageActDeAct` = 125, `errCodeTickTimeOutTimer` = 126, `errCodeOperateModeStateError` = 127,
`errCodeHeatingTempReadErrEEPROM` = 128, `errCodeMPFailedStart` = 129, `errCodeReadErrEEPROM` = 130, `errCodeTimeOutWaitingForVoltages` = 131,
`errCodeMAX` }
- enum `PowerMgrConf` { `Normal` = 0, `ApplicationControlled` = 1, `BatterySuspend` = 2 }
 - enum `PowerMgrStatus` { `NoRequestsPending` = 0, `SuspendPending` = 1, `ShutdownPending` = 2 }
 - enum `TouchScreenModeSettings` { `MOUSE_NEXT_BOOT` = 0, `TOUCH_NEXT_BOOT` = 1, `MOUSE_NOW` = 2, `TOUCH_NOW` = 3 }
 - enum `TSAdvancedSettingsParameter` {
`TS_RIGHT_CLICK_TIME` = 0, `TS_LOW_LEVEL` = 1, `TS_UNTOUCHLEVEL` = 2, `TS_DEBOUNCE_TIME` = 3,
`TS_DEBOUNCE_TIMEOUT_TIME` = 4, `TS_DOUBLECLICK_MAX_CLICK_TIME` = 5, `TS_DOUBLE_CLICK_TIME` = 6, `TS_MAX_RIGHTCLICK_DISTANCE` = 7,
`TS_USE_DEJITTER` = 8, `TS_CALIBTATION_WIDTH` = 9, `TS_CALIBRATION_MEASUREMENTS` = 10, `TS_RESTORE_DEFAULT_SETTINGS` = 11
}
 - enum `CalibrationModeSettings` {
`MODE_UNKNOWN` = 0, `MODE_NORMAL` = 1, `MODE_CALIBRATION_5P` = 2, `MODE_CALIBRATION_9P` = 3,
`MODE_CALIBRATION_13P` = 4 }
 - enum `CalibrationConfigParam` {
`CONFIG_CALIBRATION_WITH` = 0, `CONFIG_CALIBRATION_MEASUREMENTS` = 1, `CONFIG_5P_CALIBRATION_POINT_BORDER` = 2, `CONFIG_13P_CALIBRATION_POINT_BORDER` = 3,
`CONFIG_13P_CALIBRATION_TRANSITION_MIN` = 4, `CONFIG_13P_CALIBRATION_TRANSITION_MAX` = 5 }

Functions

- `EXTERN_C CCAUXDLL_API char`
`const *CCAUXDLL_CALLING_CONV GetErrorStringA (eErr errCode)`
- `EXTERN_C CCAUXDLL_API wchar_t`
`const *CCAUXDLL_CALLING_CONV GetErrorStringW (eErr errCode)`

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

Variables

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

5.1.1 Typedef Documentation

5.1.1.1 typedef enum CrossControl::PowerMgrConf _PowerMgrConf

Enumeration of the settings that can be used with the [PowerMgr](#) system.

5.1.1.2 typedef enum CrossControl::PowerMgrStatus _PowerMgrStatus

5.1.1.3 typedef struct version_info VersionType

5.1.2 Enumeration Type Documentation

5.1.2.1 enum 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.

5.1.2.2 enum CalibrationConfigParam

Touch screen caibration parameters

Enumerator

CONFIG_CALIBRATION_WITH

CONFIG_CALIBRATION_MEASUREMENTS Accepted error value when calibrating.

CONFIG_5P_CALIBRATION_POINT_BORDER Number of measurements to accept a calibration point.

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

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

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

5.1.2.3 enum CalibrationModeSettings

Touch screen caibration modes

Enumerator

MODE_UNKNOWN

MODE_NORMAL Unknown mode.

MODE_CALIBRATION_5P Normal operation mode.

MODE_CALIBRATION_9P Calibration with 5 points mode.

MODE_CALIBRATION_13P Calibration with 9 points mode.

5.1.2.4 enum CanFrameType

Can frame type settings

Enumerator

FrameStandard

FrameExtended

FrameStandardExtended

5.1.2.5 enum 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

5.1.2.6 enum CCStatus

Enable/disable enumeration

Enumerator

Disabled

Enabled The setting is disabled or turned off

5.1.2.7 enum ChargingStatus

Current charging status of the battery.

Enumerator

ChargingStatus_NoCharge The battery is not being charged. System is running on battery power.

ChargingStatus_Charging The battery is currently being charged

ChargingStatus_FullyCharged The battery is fully charged

ChargingStatus_TempLow The temperature is too low to allow the battery to be charged

ChargingStatus_TempHigh The temperature is too high to allow the battery to be charged

ChargingStatus_Unknown There was an error determining the charging status

5.1.2.8 enum 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

5.1.2.9 enum 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

5.1.2.10 enum ErrorStatus

Enumerator

ErrorStatus_NoError
ErrorStatus_ThermistorTempSensor
ErrorStatus_SecondaryTempSensor
ErrorStatus_ChargeFail
ErrorStatus_Overcurrent
ErrorStatus_Init

5.1.2.11 enum hwErrorStatusCodes

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

Enumerator

errCodeNoErr
errCodeFPGACONFReadErr
errCodeFPGACONFUnexpVal
errCodeCBRESETReadErr
errCodeSUS3ReadErr
errCodeSUS4ReadErr
errCodeSUS5ReadErr
errCodePG5VSTBYReadErr

errCodePG5VSTBYUnexpVal
errCodeCANPWROKReadErr
errCodeVIDPWROKReadErr
errCodeLVDSBLENReadErr
errCodeLVDSVDDENReadErr
errCodeEXTCTRLONReadErr
errCodeFPBTNONReadErr
errCode24VReadErr
errCode24VOutOfLimits
errCode24VINReadErr
errCode24VINOutOfLimits
errCode12VReadErr
errCode12VOutOfLimits
errCode12VVIDEORReadErr
errCode12VVIDEOOutOfLimits
errCode5VSTBYReadErr
errCode5VSTBYOutOfLimits
errCode5VReadErr
errCode5VOutOfLimits
errCode3V3ReadErr
errCode3V3OutOfLimits
errCodeTFTVOLReadErr
errCodeTFTVLOOutOfLimits
errCode1V9ReadErr
errCode1V9OutOfLimits
errCode1V8ReadErr
errCode1V8OutOfLimits
errCode1V5ReadErr
errCode1V5OutOfLimits
errCode1V2ReadErr
errCode1V2OutOfLimits
errCode1V05ReadErr
errCode1V05OutOfLimits
errCode1V0ReadErr
errCode1V0OutOfLimits
errCode0V9ReadErr
errCode0V9OutOfLimits
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
errCodeFPBTNUnexpVal
errCodeEXTCTRLUnexpVal
errCodeMAINPWROKReadErr
errCodeFRONTSPAREReadErr
errCodeTIMERReadErr
errCodeManageDiagnostics
errCodeFPBTNTimOutReadErrEEPROM
errCodeEXTCTRLTimOutReadErrEEPROM

errCodeFPBTNAndExtCtrlDisabled
errCodeSWVerReadErr
errCodeSWVerWriteErr
errCodeManageActDeAct
errCodeTickTimeOutTimer
errCodeOperateModeStateError
errCodeHeatingTempReadErrEEPROM
errCodeMPFailedStart
errCodeReadErrEEPROM
errCodeTimeOutWaitingForVoltages
errCodeMAX

5.1.2.12 enum JidaSensorType

Jida temperature sensor types

Enumerator

TEMP_CPU
TEMP_BOX
TEMP_ENV
TEMP_BOARD
TEMP_BACKPLANE
TEMP_CHIPSETS
TEMP_VIDEO
TEMP_OTHER

5.1.2.13 enum LightSensorOperationRange

Light sensor operation ranges.

Enumerator

RangeStandard
RangeExtended Light sensor operation range standard

5.1.2.14 enum 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.

5.1.2.15 enum PowerAction

Button and on/off signal actions.

Enumerator

NoAction No action taken

ActionSuspend The system enters suspend mode

ActionShutDown The system shuts down

5.1.2.16 enum PowerMgrConf

Enumeration of the settings that can be used with the [PowerMgr](#) system.

Enumerator

Normal Applications will not be able to delay suspend/shutdown requests. This is the normal configuration that is used when the [PowerMgr](#) class is not being used. Setting this configuration turns off the feature if it is in use.

ApplicationControlled Applications can delay suspend/shutdown requests.

BatterySuspend In this mode, the computer will automatically enter suspend mode when the unit starts running on battery power. Applications can delay suspend/shutdown requests. This mode is only applicable if the unit has an external battery. Using this configuration on a computer without an external battery will be the same as using the configuration `ApplicationControlled`.

5.1.2.17 enum PowerMgrStatus

Enumerator

NoRequestsPending No suspend or shutdown requests.

SuspendPending A suspend request is pending.

ShutdownPending A shutdown request is pending.

5.1.2.18 enum PowerSource

Current power source of the computer.

Enumerator

PowerSource_Battery

PowerSource_ExternalPower

5.1.2.19 enum shutdownReasonCodes

The shutdown codes returned by getShutdownReason.

Enumerator

shutdownReasonCodeNoError

5.1.2.20 enum 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

5.1.2.21 enum 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.

5.1.2.22 enum 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

5.1.2.23 enum 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.

5.1.2.24 enum 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

5.1.2.25 enum VideoChannel

The available analog video channels

Enumerator

Analog_Channel_1

Analog_Channel_2

Analog_Channel_3

Analog_Channel_4

5.1.2.26 enum 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

5.1.2.27 enum 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

5.1.3 Function Documentation

5.1.3.1 **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.

5.1.3.2 **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.

5.1.3.3 **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.

**5.1.3.4 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.

**5.1.3.5 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.

**5.1.3.6 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.

5.1.4 Variable Documentation

5.1.4.1 `const unsigned char DigitalIn_1 = (1 << 0)`

Bit defines for getDigIO

5.1.4.2 `const unsigned char DigitalIn_2 = (1 << 1)`

5.1.4.3 `const unsigned char DigitalIn_3 = (1 << 2)`

5.1.4.4 `const unsigned char DigitalIn_4 = (1 << 3)`

5.1.4.5 `const unsigned char Video1Conf = (1 << 0)`

Bit defines for getVideoStartupPowerConfig and setVideoStartupPowerConfig

5.1.4.6 `const unsigned char Video2Conf = (1 << 1)`

[Video](#) channel 1 config

5.1.4.7 `const unsigned char Video3Conf = (1 << 2)`

[Video](#) channel 2 config

5.1.4.8 `const unsigned char Video4Conf = (1 << 3)`

[Video](#) channel 3 config

Chapter 6

Data Structure Documentation

6.1 About Struct Reference

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

Public Member Functions

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

6.1.1 Detailed Description

Get information about the CCpilot XM computer.

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

Example Usage:

```

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

using namespace std;

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

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

    if(!buffer)
        return;

    CrossControl::eErr err;

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

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

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

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

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

bool isGPSSMounted;
err = pAbout->getIsGPSSMounted (&isGPSSMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "GPS mounted: " << (isGPSSMounted ? "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();
}

```

6.1.2 Member Function Documentation

6.1.2.1 virtual eErr 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;
```

6.1.2.2 virtual eErr 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;
```

6.1.2.3 virtual eErr 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;
```

6.1.2.4 virtual eErr 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;
```

6.1.2.5 virtual eErr 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;
```

6.1.2.6 virtual eErr getIsDisplayAvailable (bool * *available*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.7 virtual eErr 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;
```

6.1.2.8 virtual eErr 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;
```

6.1.2.9 virtual eErr getIsTouchScreenAvailable (bool * *available*) [pure virtual]

Get Display [TouchScreen](#) status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.10 virtual eErr 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;
```

6.1.2.11 virtual eErr 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;
```

6.1.2.12 virtual eErr 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;
```

6.1.2.13 virtual eErr 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;
```

6.1.2.14 virtual eErr 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;
```

6.1.2.15 virtual eErr 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;
```

6.1.2.16 virtual eErr 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;
```

6.1.2.17 virtual eErr getNrOfCANConnections (unsigned char * NrOfConnections) [pure virtual]

Get number of CAN connections present.

Parameters

<i>NrOfConnections</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;
```

6.1.2.18 virtual eErr getNrOfDigIOConnections (unsigned char * *NrOfConnections*)
[pure virtual]

Get number of digital I/O connections present.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.19 virtual eErr 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;
```

6.1.2.20 virtual eErr 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;
```

6.1.2.21 virtual eErr 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;
```

6.1.2.22 virtual eErr 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;
```

6.1.2.23 virtual eErr 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;
```

6.1.2.24 virtual void Release () [pure virtual]

Delete the [About](#) object.

Returns

-

Example Usage:

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

list_about_information (pAbout);

pAbout->Release ();
```

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

- IncludeFiles/[About.h](#)

6.2 Adc Struct Reference

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

Public Member Functions

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

6.2.1 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();
}

```

6.2.2 Member Function Documentation

6.2.2.1 virtual CrossControl::eErr 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;
}
```

6.2.2.2 virtual void 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:

- IncludeFiles/[Adc.h](#)

6.3 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

6.3.1 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();
}

```

6.3.2 Member Function Documentation

6.3.2.1 virtual eErr 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;

```

6.3.2.2 virtual eErr 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;
```

6.3.2.3 virtual eErr 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;

```

6.3.2.4 virtual eErr 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;

```

6.3.2.5 virtual eErr 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;
```

6.3.2.6 virtual eErr 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;
```

6.3.2.7 virtual void Release () [pure virtual]

Delete the [AuxVersion](#) object.

Returns

-

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

- IncludeFiles/[AuxVersion.h](#)

6.4 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

6.4.1 Detailed Description**Backlight settings**

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

Example Usage:

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

```

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

    CrossControl::eErr err;
    unsigned char value;

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

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

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

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

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

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

}

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

    change_backlight (pBacklight);

    pBacklight->Release ();
}

```

6.4.2 Member Function Documentation

6.4.2.1 `virtual eErr 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>averageWndSize</i>	The average window size in nr of samples.
<i>rejectWndSize</i>	The reject window size in nr of samples.
<i>rejectDeltaInLux</i>	The reject delta in lux.
<i>mode</i>	The configured sampling mode.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.4.2.2 `virtual eErr getAutomaticBLParams (bool * bSoftTransitions, double * k)` [pure virtual]

Get parameters for automatic backlight control.

Parameters

<i>bSoftTransitions</i>	Soft transitions used?
<i>k</i>	K value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.4.2.3 `virtual eErr 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.

6.4.2.4 virtual eErr getIntensity (unsigned char * *intensity*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

6.4.2.5 virtual eErr 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.

6.4.2.6 virtual eErr getStatus (unsigned char * *status*) [pure virtual]

Get backlight controller status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.4.2.7 virtual void Release () [pure virtual]

Delete the backlight object.

Returns

-

Example Usage:

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

change_backlight(pBacklight);

pBacklight->Release();
```

6.4.2.8 virtual eErr 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.

6.4.2.9 virtual eErr setAutomaticBLParams (bool *bSoftTransitions*) [pure virtual]

Set parameters for automatic backlight control.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.4.2.10 virtual eErr setIntensity (unsigned char *intensity*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

6.4.2.11 virtual eErr 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.

6.4.2.12 virtual eErr 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.

6.4.2.13 virtual eErr 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:

- IncludeFiles/[Backlight.h](#)

6.5 Battery Struct Reference

```
#include <Battery.h>
```

Public Member Functions

- virtual eErr [isBatteryPresent](#) (bool *batteryIsPresent)=0
- virtual eErr [getBatteryVoltageStatus](#) (unsigned char *batteryVoltagePercent)=0
- virtual eErr [getBatteryChargingStatus](#) ([ChargingStatus](#) *status)=0
- virtual eErr [getPowerSource](#) ([PowerSource](#) *status)=0
- virtual eErr [getBatteryTemp](#) (signed short *temperature)=0
- virtual eErr [getHwErrorStatus](#) ([ErrorStatus](#) *errorCode)=0
- virtual eErr [getTimer](#) ([BatteryTimerType](#) *times)=0
- virtual eErr [getMinMaxTemp](#) (signed short *minTemp, signed short *maxTemp)=0
- virtual eErr [getBatteryHWversion](#) (char *buff, int len)=0
- virtual eErr [getBatterySwVersion](#) (unsigned short *major, unsigned short *minor, unsigned short *release, unsigned short *build)=0
- virtual eErr [getBatterySerial](#) (char *buff, int len)=0
- virtual void [Release](#) ()=0

6.5.1 Detailed Description

External battery status and settings (Only available on specific models)

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

Example Usage:

```

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

using namespace std;
using namespace CrossControl;

void readBatteryInfo(BATTERYHANDLE pBattery)
{
    eErr error;
    bool bpresent;
    std::string input;

    if(!pBattery)
        return;

    error = pBattery->isBatteryPresent (&bpresent);

    if(error != ERR_SUCCESS)
    {
        cout << "isBatteryPresent: " << GetErrorStringA(error) << std::endl;
    }
    else
    {
        if(bpresent)
        {
            cout << " Battery is present. Testing functionality... " << std::endl;
        }
        else
        {
            cout << " Battery is NOT present." << std::endl;
        }
    }

    ChargingStatus cs;
    error = pBattery->getBatteryChargingStatus (&cs);
    if(error == ERR_NOT_SUPPORTED && !bpresent)
    {
        cout << "getBatteryChargingStatus: " << GetErrorStringA(error) << " - Ok, since battery
            is not present!" << std::endl;
    }
    else if(error != ERR_SUCCESS)
    {
        cout << "getBatteryChargingStatus: " << GetErrorStringA(error) << std::endl;
    }
    else
    {
        switch(cs)
        {
            case ChargingStatus_NoCharge:
                cout << "getBatteryChargingStatus: Battery is not being charged" << std::endl;
                break;
            case ChargingStatus_Charging:
                cout << "getBatteryChargingStatus: Battery is being charged" << std::endl;
                break;
            case ChargingStatus_FullyCharged:
                cout << "getBatteryChargingStatus: Battery is fully charged" << std::endl;
                break;
            case ChargingStatus_TempLow:

```

```

        cout << "getBatteryChargingStatus: Temperature is too low to charge the battery" << std::endl;
        break;
    case ChargingStatus_TempHigh:
        cout << "getBatteryChargingStatus: Temperature is too high to charge the battery" << std::endl;
        break;
    case ChargingStatus_Unknown:
        cout << "getBatteryChargingStatus: ChargingStatus_Unknown" << std::endl;
        break;
    default:
        cout << "getBatteryChargingStatus: invalid return value" << std::endl;
        break;
    }
}

char buf[255];
error = pBattery->getBatteryHWVersion(buf, sizeof(buf));
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryHWVersion: " << GetErrorStringA(error) << " - Ok, since battery is
        not present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatteryHWVersion: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatteryHWVersion: " << buf << std::endl;
}

error = pBattery->getBatterySerial(buf, sizeof(buf));
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatterySerial: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatterySerial: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatterySerial: " << buf << std::endl;
}

unsigned short major;
unsigned short minor;
unsigned short release;
unsigned short build;
error = pBattery->getBatterySwVersion(&major, &minor, &release, &build);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatterySwVersion: " << GetErrorStringA(error) << " - Ok, since battery is
        not present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatterySwVersion: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatterySwVersion: v" << major << "." << minor << "." << release << "." << build <<
        std::endl;
}

short temp;
error = pBattery->getBatteryTemp(&temp);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryTemp: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}

```

```

}
else if(error != ERR_SUCCESS)
{
    cout << "getBatteryTemp: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatteryTemp: " << temp << " deg C" << std::endl;
}

unsigned char s;
error = pBattery->getBatteryVoltageStatus(&s);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryVoltageStatus: " << GetErrorStringA(error) << " - Ok, since battery
        is not present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatteryVoltageStatus: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatteryVoltageStatus: " << (int)s << " %" << std::endl;
}

ErrorStatus es;
error = pBattery->getHwErrorStatus(&es);

if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getHwErrorStatus: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getHwErrorStatus: " << GetErrorStringA(error) << std::endl;
}
else
{
    switch(es)
    {
        case ErrorStatus_NoError:
            cout << "getHwErrorStatus: " << "Battery reports no HW errors" << std::endl;
            break;
        case ErrorStatus_ThermistorTempSensor:
            cout << "getHwErrorStatus: " << "Battery error! The thermistor temp sensor is not working" <<
                std::endl;
            break;
        case ErrorStatus_SecondaryTempSensor:
            cout << "getHwErrorStatus: " << "Battery error! The secondary temp sensor is not working" <<
                std::endl;
            break;
        case ErrorStatus_ChargeFail:
            cout << "getHwErrorStatus: " << "Battery error! Charging failed" << std::endl;
            break;
        case ErrorStatus_Overcurrent:
            cout << "getHwErrorStatus: " << "Battery error! Overcurrent detected" << std::endl;
            break;
        case ErrorStatus_Init:
            cout << "getHwErrorStatus: " << "Battery error! Battery not initiated" << std::endl;
            break;
        default:
            cout << "getHwErrorStatus: " << "invalid return value" << std::endl;
            break;
    }
}

short max;
error = pBattery->getMinMaxTemp(&temp, &max);

```

```

if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getMinMaxTemp: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getMinMaxTemp: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getMinMaxTemp: MinTemp:" << temp << ", MaxTemp: " << max << std::endl;
}
}

PowerSource ps;
error = pBattery->getPowerSource(&ps);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getPowerSource: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getPowerSource: " << GetErrorStringA(error) << std::endl;
}
else
{
    if(ps == PowerSource_Battery)
        cout << "getPowerSource: Power source: Battery" << std::endl;
    else
        cout << "getPowerSource: Power source: External Power" << std::endl;
}
}

BatteryTimerType times;
memset(&times, 0, sizeof(times));
error = pBattery->getTimer(&times);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getTimer: " << GetErrorStringA(error) << " - Ok, since battery is not present!"
        << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getTimer: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getTimer: " << std::endl;
    cout << "Total run time on main power=" << times.TotRunTimeMain*60 << " min(s)" <<
        std::endl
        << "Total run time on battery power=" << times.TotRunTimeBattery*60 << " min(s)" <<
        std::endl
        << "Total run time below -20C=" << times.RunTime_m20 << " min(s)" << std::endl
        << "Total run time -20-0C=" << times.RunTime_m20_0 << " min(s)" << std::endl
        << "Total run time 0-40C=" << times.RunTime_0_40 << " min(s)" << std::endl
        << "Total run time 40-60C=" << times.RunTime_40_60 << " min(s)" << std::endl
        << "Total run time 60-70C=" << times.RunTime_60_70 << " min(s)" << std::endl
        << "Total run time 70-80C=" << times.RunTime_70_80 << " min(s)" << std::endl
        << "Total run time above 80C=" << times.RunTime_Above80 << " min(s)" << std::endl;
}
}

int main(void)
{
    BATTERYHANDLE pBattery = ::GetBattery();
    assert(pBattery);

    readBatteryInfo(pBattery);

    pBattery->Release();
}

```

}

6.5.2 Member Function Documentation

6.5.2.1 virtual eErr getBatteryChargingStatus (ChargingStatus * *status*) [pure virtual]

Get battery charging status.

Parameters

<i>status</i>	the current charging mode of the battery.
---------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

ChargingStatus cs;
error = pBattery->getBatteryChargingStatus(&cs);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryChargingStatus: " << GetErrorStringA(error) << " - Ok, since battery
        is not present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatteryChargingStatus: " << GetErrorStringA(error) << std::endl;
}
else
{
    switch(cs)
    {
        case ChargingStatus_NoCharge:
            cout << "getBatteryChargingStatus: Battery is not being charged" << std::endl;
            break;
        case ChargingStatus_Charging:
            cout << "getBatteryChargingStatus: Battery is being charged" << std::endl;
            break;
        case ChargingStatus_FullyCharged:
            cout << "getBatteryChargingStatus: Battery is fully charged" << std::endl;
            break;
        case ChargingStatus_TempLow:
            cout << "getBatteryChargingStatus: Temperature is too low to charge the battery" << std::endl;
            break;
        case ChargingStatus_TempHigh:
            cout << "getBatteryChargingStatus: Temperature is too high to charge the battery" << std::endl;
            break;
        case ChargingStatus_Unknown:
            cout << "getBatteryChargingStatus: ChargingStatus_Unknown" << std::endl;
            break;
        default:
            cout << "getBatteryChargingStatus: invalid return value" << std::endl;
            break;
    }
}

```

6.5.2.2 virtual eErr getBatteryHWversion (char * *buff*, int *len*) [pure virtual]

Get battery 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:

```
char buf[255];
error = pBattery->getBatteryHWversion(buf, sizeof(buf));
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryHWversion: " << GetErrorStringA(error) << " - Ok, since battery is
        not present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatteryHWversion: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatteryHWversion: " << buf << std::endl;
}
}
```

6.5.2.3 virtual eErr getBatterySerial (char * buff, int len) [pure virtual]

Get battery 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. The serial number is 10 characters plus terminating zero, in total 11 bytes in size.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
error = pBattery->getBatterySerial(buf, sizeof(buf));
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatterySerial: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatterySerial: " << GetErrorStringA(error) << std::endl;
}
else
{
}
```

```

    cout << "getBatterySerial: " << buf << std::endl;
}

```

6.5.2.4 virtual eErr getBatterySwVersion (unsigned short * *major*, unsigned short * *minor*, unsigned short * *release*, unsigned short * *build*) [pure virtual]

Get the battery 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:

```

unsigned short major;
unsigned short minor;
unsigned short release;
unsigned short build;
error = pBattery->getBatterySwVersion(&major, &minor, &release, &build);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatterySwVersion: " << GetErrorStringA(error) << " - Ok, since battery is
        not present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatterySwVersion: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatterySwVersion: v" << major << "." << minor << "." << release << "." << build <<
        std::endl;
}

```

6.5.2.5 virtual eErr getBatteryTemp (signed short * *temperature*) [pure virtual]

Get battery temperature.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
short temp;
error = pBattery->getBatteryTemp(&temp);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryTemp: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatteryTemp: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatteryTemp: " << temp << " deg C" << std::endl;
}
}
```

6.5.2.6 virtual eErr getBatteryVoltageStatus (unsigned char * *batteryVoltagePercent*) [pure virtual]

Get battery voltage status.

Parameters

<i>battery-Voltage-Percent</i>	the current voltage level of the battery, in percent [0..100].
--------------------------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char s;
error = pBattery->getBatteryVoltageStatus(&s);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryVoltageStatus: " << GetErrorStringA(error) << " - Ok, since battery
        is not present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getBatteryVoltageStatus: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getBatteryVoltageStatus: " << (int)s << " %" << std::endl;
}
}
```

6.5.2.7 virtual eErr getHwErrorStatus (ErrorStatus * *errorCode*) [pure virtual]

Get hardware error code. If hardware errors are found or other problems are discovered by the battery pack, they are reported here.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

ErrorStatus es;
error = pBattery->getHwErrorStatus(&es);

if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getHwErrorStatus: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getHwErrorStatus: " << GetErrorStringA(error) << std::endl;
}
else
{
    switch(es)
    {
        case ErrorStatus_NoError:
            cout << "getHwErrorStatus: " << "Battery reports no HW errors" << std::endl;
            break;
        case ErrorStatus_ThermistorTempSensor:
            cout << "getHwErrorStatus: " << "Battery error! The thermistor temp sensor is not working" <<
                std::endl;
            break;
        case ErrorStatus_SecondaryTempSensor:
            cout << "getHwErrorStatus: " << "Battery error! The secondary temp sensor is not working" <<
                std::endl;
            break;
        case ErrorStatus_ChargeFail:
            cout << "getHwErrorStatus: " << "Battery error! Charging failed" << std::endl;
            break;
        case ErrorStatus_Overcurrent:
            cout << "getHwErrorStatus: " << "Battery error! Overcurrent detected" << std::endl;
            break;
        case ErrorStatus_Init:
            cout << "getHwErrorStatus: " << "Battery error! Battery not initiated" << std::endl;
            break;
        default:
            cout << "getHwErrorStatus: " << "invalid return value" << std::endl;
            break;
    }
}

```

6.5.2.8 virtual eErr getMinMaxTemp (signed short * *minTemp*, signed short * *maxTemp*) [pure virtual]

Get temperature interval of the battery.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
short max;
error = pBattery->getMinMaxTemp(&temp, &max);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getMinMaxTemp: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getMinMaxTemp: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getMinMaxTemp: MinTemp:" << temp << ", MaxTemp: " << max << std::endl;
}
}
```

6.5.2.9 virtual eErr getPowerSource (PowerSource * status) [pure virtual]

Get the currently used power source.

Parameters

<i>status</i>	the current power source, external power or battery.
---------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
PowerSource ps;
error = pBattery->getPowerSource(&ps);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getPowerSource: " << GetErrorStringA(error) << " - Ok, since battery is not
        present!" << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getPowerSource: " << GetErrorStringA(error) << std::endl;
}
else
{
    if(ps == PowerSource_Battery)
        cout << "getPowerSource: Power source: Battery" << std::endl;
    else
        cout << "getPowerSource: Power source: External Power" << std::endl;
}
}
```

6.5.2.10 virtual eErr getTimer (BatteryTimerType * *times*) [pure virtual]

Get battery diagnostic timer.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
BatteryTimerType times;
memset(&times, 0, sizeof(times));
error = pBattery->getTimer(&times);
if(error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getTimer: " << GetErrorStringA(error) << " - Ok, since battery is not present!"
        << std::endl;
}
else if(error != ERR_SUCCESS)
{
    cout << "getTimer: " << GetErrorStringA(error) << std::endl;
}
else
{
    cout << "getTimer: " << std::endl;
    cout << "Total run time on main power=" << times.TotRunTimeMain*60 << " min(s)" << std::endl
        << "Total run time on battery power=" << times.TotRunTimeBattery*60 << " min(s)" << std::endl
        << "Total run time below -20C=" << times.RunTime_m20 << " min(s)" << std::endl
        << "Total run time -20-0C=" << times.RunTime_m20_0 << " min(s)" << std::endl
        << "Total run time 0-40C=" << times.RunTime_0_40 << " min(s)" << std::endl
        << "Total run time 40-60C=" << times.RunTime_40_60 << " min(s)" << std::endl
        << "Total run time 60-70C=" << times.RunTime_60_70 << " min(s)" << std::endl
        << "Total run time 70-80C=" << times.RunTime_70_80 << " min(s)" << std::endl
        << "Total run time above 80C=" << times.RunTime_Above80 << " min(s)" << std::endl;
}
}
```

6.5.2.11 virtual eErr isBatteryPresent (bool * *batteryIsPresent*) [pure virtual]

Is an external battery connected?

Parameters

<i>batteryIsPresent</i>	true if a battery is connected, otherwise false.
-------------------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
error = pBattery->isBatteryPresent(&bpresent);
```

```

if(error != ERR_SUCCESS)
{
    cout << "isBatteryPresent: " << GetErrorStringA(error) << std::endl;
}
else
{
    if(bpresent)
    {
        cout << " Battery is present. Testing functionality... " << std::endl;
    }
    else
    {
        cout << " Battery is NOT present." << std::endl;
    }
}
}

```

6.5.2.12 virtual void Release () [pure virtual]

Delete the [Battery](#) object.

Returns

-

Example Usage:

```

BATTERYHANDLE pBattery = ::GetBattery();
assert(pBattery);

readBatteryInfo(pBattery);

pBattery->Release();

```

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

- IncludeFiles/[Battery.h](#)

6.6 BatteryTimerType Struct Reference

```
#include <Battery.h>
```

Data Fields

- unsigned long [TotRunTimeMain](#)
- unsigned long [TotRunTimeBattery](#)
- unsigned long [RunTime_m20](#)
- unsigned long [RunTime_m20_0](#)
- unsigned long [RunTime_0_40](#)
- unsigned long [RunTime_40_60](#)
- unsigned long [RunTime_60_70](#)
- unsigned long [RunTime_70_80](#)
- unsigned long [RunTime_Above80](#)

6.6.1 Detailed Description

[Diagnostic](#) timer data

6.6.2 Field Documentation**6.6.2.1 unsigned long RunTime_0_40**

Total runtime in range 0 to -20 deg C (minutes)

6.6.2.2 unsigned long RunTime_40_60

Total runtime in range 0 to 40 deg C (minutes)

6.6.2.3 unsigned long RunTime_60_70

Total runtime in range 40 to 60 deg C (minutes)

6.6.2.4 unsigned long RunTime_70_80

Total runtime in range 60 to 70 deg C (minutes)

6.6.2.5 unsigned long RunTime_Above80

Total runtime in range 70 to 80 deg C (minutes)

6.6.2.6 unsigned long RunTime_m20

Total running time on battery power (minutes)

6.6.2.7 unsigned long RunTime_m20_0

Total runtime below -20 deg C (minutes)

6.6.2.8 unsigned long TotRunTimeBattery

Total running time on main power (minutes)

6.6.2.9 unsigned long TotRunTimeMain

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

- IncludeFiles/[Battery.h](#)

6.7 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

6.7.1 Detailed Description

Buzzer settings

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

Example Usage:

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

using namespace std;
using namespace CrossControl;

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

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

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

```

        return;

    CrossControl::eErr err;
    unsigned short vol;

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

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

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

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

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

    play_beeps(pBuzzer);

    pBuzzer->Release();
}

```

6.7.2 Member Function Documentation

6.7.2.1 virtual eErr buzze (int *time*, bool *blocking*) [pure virtual]

Buzzes for a specified time.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pBuzzer->setFrequency(freq);
```

```

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

```

6.7.2.2 virtual eErr 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.

6.7.2.3 virtual eErr 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.

6.7.2.4 virtual eErr getVolume (unsigned short * *volume*) [pure virtual]

Get buzzer volume.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

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

```

6.7.2.5 virtual void Release () [pure virtual]

Delete the [Buzzer](#) object.

Returns

-

Example Usage:

```

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

play_beeps(pBuzzer);

pBuzzer->Release();

```

6.7.2.6 virtual eErr setFrequency (unsigned short *frequency*) [pure virtual]

Set buzzer frequency.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

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

```

6.7.2.7 virtual eErr 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.

6.7.2.8 virtual eErr setVolume (unsigned short *volume*) [pure virtual]

Set buzzer volume.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

- IncludeFiles/[Buzzer.h](#)

6.8 BuzzerSetup Struct Reference

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

Data Fields

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

6.8.1 Field Documentation

6.8.1.1 unsigned short frequency

buzzer frequency

6.8.1.2 unsigned short volume

buzzer volume

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

- [IncludeFiles/CCAuxTypes.h](#)

6.9 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

6.9.1 Detailed Description

Can settings

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

Example Usage:

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

using namespace std;
using namespace CrossControl;

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

```

if(!pCanSetting)
    return;

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

#ifdef LINUX

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

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

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

    read_cansettings(pCanSetting);

    pCanSetting->Release();
}

```

6.9.2 Member Function Documentation

6.9.2.1 virtual eErr getBaudrate (unsigned char *net*, unsigned short * *baudrate*) [pure virtual]

Get Baud rate

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.9.2.2 virtual eErr getFrameType (unsigned char *net*, CanFrameType * *frameType*) [pure virtual]

Get frame type**Parameters**

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.9.2.3 virtual void Release () [pure virtual]

Delete the [CanSetting](#) object.

Returns

-

Example Usage:

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

read_cansettings(pCanSetting);

pCanSetting->Release();
```

6.9.2.4 virtual eErr 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.

6.9.2.5 virtual eErr 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:

- IncludeFiles/[CanSetting.h](#)

6.10 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

6.10.1 Detailed Description

Video channel 4 config

Configuration of various settings

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

Example Usage:

```

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

using namespace std;
using namespace CrossControl;

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

    if(!pConfig)
        return;

    err = pConfig->getStartupTriggerConfig(&trig);
    if(err == ERR_SUCCESS)
    {
        cout << "Start-up trigger is set to: ";
    }
}

```

```

switch(trig)
{
case Front_Button_Enabled: cout << "Front button only" << endl; break;
case OnOff_Signal_Enabled: cout << "On/Off signal only" << endl; break;
case Both_Button_And_Signal_Enabled: cout << "Front button or On/off
signal" << endl; break;
default: cout << "Error - Undefined StartupTrigger" << endl; break;
}
}
else
{
cout << "Error(" << err << ") in function getStartupTriggerConfig: " <<
GetErrorStringA(err) << endl;
}

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

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

conf_example(pConfig);

pConfig->Release();
}

```

6.10.2 Member Function Documentation

6.10.2.1 virtual eErr 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.

6.10.2.2 `virtual eErr 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.

6.10.2.3 `virtual eErr 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.

6.10.2.4 `virtual eErr 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.

6.10.2.5 `virtual eErr 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.

6.10.2.6 `virtual eErr 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.

6.10.2.7 `virtual eErr 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.

6.10.2.8 virtual eErr 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.

6.10.2.9 virtual eErr 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.

6.10.2.10 virtual eErr getStartupTriggerConfig (TriggerConf * *configuration*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.10.2.11 virtual eErr 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.

6.10.2.12 virtual eErr 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.

6.10.2.13 virtual eErr 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.

6.10.2.14 virtual void Release () [pure virtual]

Delete the [Config](#) object.

Returns

-

Example Usage:

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

conf_example(pConfig);

pConfig->Release();
```

6.10.2.15 virtual eErr 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.

6.10.2.16 `virtual eErr 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.

6.10.2.17 `virtual eErr 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.

6.10.2.18 `virtual eErr 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.

6.10.2.19 `virtual eErr 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.

6.10.2.20 `virtual eErr 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.

6.10.2.21 `virtual eErr 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.

6.10.2.22 virtual eErr 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.

6.10.2.23 virtual eErr setShortButtonPressAction (PowerAction *action*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.10.2.24 `virtual eErr 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.

6.10.2.25 `virtual eErr 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.

6.10.2.26 `virtual eErr 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.

6.10.2.27 `virtual eErr 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:

- IncludeFiles/[Config.h](#)

6.11 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

6.11.1 Detailed Description

Access to unit diagnostic data

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

Example Usage:

```

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

using namespace std;
using namespace CrossControl;

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

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

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

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

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

    if(!pDiagnostic)
        return;

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

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

```

```

    printStringTime(err, "Total run time 70-80 deg C", tt.RunTime70_80);
    printStringTime(err, "Total run time above 80 deg C", tt.Above80RunTime);

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

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

    diagnostic_example(pDiagnostic);

    pDiagnostic->Release();
}

```

6.11.2 Member Function Documentation

6.11.2.1 virtual eErr 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.

6.11.2.2 virtual eErr getHwErrorStatus (unsigned short * *errorCode*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.11.2.3 virtual eErr getMinMaxTemp (signed short * *minTemp*, signed short * *maxTemp*) [pure virtual]

Get diagnostic temperature interval of the unit.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.11.2.4 virtual eErr 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.

6.11.2.5 virtual eErr 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.

6.11.2.6 virtual eErr 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.

6.11.2.7 `virtual eErr 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.

6.11.2.8 `virtual eErr getSSTemp (signed short * temperature)` [pure virtual]

Get System Supervisor temperature.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.11.2.9 `virtual eErr 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.

6.11.2.10 virtual eErr getTimer (TimerType * times) [pure virtual]

Get diagnostic timer.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.11.2.11 virtual void Release () [pure virtual]

Delete the [Diagnostic](#) object.

Returns

-

Example Usage:

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

diagnostic_example(pDiagnostic);

pDiagnostic->Release();
```

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

- IncludeFiles/[Diagnostic.h](#)

6.12 DigIO Struct Reference

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

Public Member Functions

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

6.12.1 Detailed Description

Read digital inputs

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

Example Usage:

```

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

using namespace std;

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

    CrossControl::eErr err;
    unsigned char inputs;

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

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

    list_digital_inputs(pDigIO);

    pDigIO->Release();
}

```

6.12.2 Member Function Documentation

6.12.2.1 virtual eErr 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;
}
```

6.12.2.2 virtual eErr getDigIODir (unsigned char * *dir*) [pure virtual]

Get Digital io direction.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

6.12.2.3 virtual void Release () [pure virtual]

Delete the [DigIO](#) object.

Returns

-

Example Usage:

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

list_digital_inputs(pDigIO);

pDigIO->Release();
```

6.12.2.4 virtual eErr setDigIO (unsigned char *state*) [pure virtual]

Set Digital outputs.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

6.12.2.5 virtual eErr setDigIODir (unsigned char *dir*) [pure virtual]

Set Digital io direction.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

- IncludeFiles/DigIO.h

6.13 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

6.13.1 Detailed Description

Firmware upgrade of the system's microprocessors and FPGA.

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

Example Usage:

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

using namespace std;

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

```

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

cout << "Upgrading SS" << endl;

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

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

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

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

upgrade->Release();
}

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

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

    cout << "Upgrading front" << endl;

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

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

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

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

    upgrade->Release();
}

```

```

}

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

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

    cout << "Upgrading FPGA" << endl;

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

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

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

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

    upgrade->Release();
}

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

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

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

    return 0;
}

```

6.13.2 Member Function Documentation

6.13.2.1 virtual eErr 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.

6.13.2.2 virtual void Release () [pure virtual]

Delete the [FirmwareUpgrade](#) object.

Returns

-

Example Usage:

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

6.13.2.3 virtual eErr shutDown () [pure virtual]

Shut down the operating system.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.13.2.4 virtual eErr startFpgaUpgrade (const char * *filename*, bool *blocking*) [pure virtual]

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

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

Parameters

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

Returns

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

Example Usage:

```

cout << "Upgrading FPGA" << endl;

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

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

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

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

```

6.13.2.5 virtual eErr 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;
        }
    }
}

```

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

Returns

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

Example Usage:

```

cout << "Upgrading front" << endl;

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

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

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

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

```

6.13.2.7 virtual eErr 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 <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 front" << endl;

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

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

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

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

```

6.13.2.8 virtual eErr startSSUpgrade (const char * filename, bool blocking) [pure virtual]

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

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

Parameters

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

Returns

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

Example Usage:

```

cout << "Upgrading SS" << endl;

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

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

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

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

```

6.13.2.9 virtual eErr startSSVerification (const char * *filename*, bool *blocking*) [pure virtual]

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

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

Parameters

<i>filename</i>	Path and filename to the .hex file to verify against.
<i>blocking</i>	Whether or not the function should wait until completion. If blocking is set to false, the function will return immediately. One must then call <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 SS" << endl;

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

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

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

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

```

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

- [IncludeFiles/FirmwareUpgrade.h](#)

6.14 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

6.14.1 Detailed Description

Front LED control

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

Example Usage:

```

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

using namespace std;
using namespace CrossControl;

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

    if(!pFrontLED)
        return;

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

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

    err = pFrontLED->getColor(&red, &green, &blue);
    if(err != ERR_SUCCESS)
    {

```

```

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

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

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

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

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

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

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

//
// Restore settings
//

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

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

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

    led_example(pFrontLED);

    pFrontLED->Release();
}

```

6.14.2 Member Function Documentation

6.14.2.1 virtual eErr getColor (unsigned char * *red*, unsigned char * *green*, unsigned char * *blue*) [pure virtual]

Get front LED color mix.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.14.2.2 virtual eErr 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.

6.14.2.3 virtual eErr 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.

6.14.2.4 virtual eErr 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.

6.14.2.5 virtual eErr 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.

6.14.2.6 virtual eErr 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.

6.14.2.7 virtual eErr 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.

6.14.2.8 virtual eErr getSignal (double * *frequency*, unsigned char * *dutyCycle*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.14.2.9 virtual void Release () [pure virtual]

Delete the [FrontLED](#) object.

Returns

-

Example Usage:

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

led_example(pFrontLED);

pFrontLED->Release();
```

6.14.2.10 virtual eErr setColor (unsigned char *red*, unsigned char *green*, unsigned char *blue*) [pure virtual]

Set front LED color mix.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.14.2.11 virtual eErr setColor (CCAuxColor color) [pure virtual]

Set one of the front LED standard colors.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.14.2.12 virtual eErr 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.

6.14.2.13 virtual eErr 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.

6.14.2.14 virtual eErr 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.

6.14.2.15 virtual eErr setOff () [pure virtual]

Set front LED off.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.16 virtual eErr setOffTime (unsigned char *offTime*) [pure virtual]

Set front LED off time.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.14.2.17 virtual eErr setOnTime (unsigned char *onTime*) [pure virtual]

Set front LED on time.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.14.2.18 virtual eErr setSignal (double *frequency*, unsigned char *dutyCycle*) [pure virtual]

Set front LED signal.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

- IncludeFiles/[FrontLED.h](#)

6.15 LedColorMixType Struct Reference

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

Data Fields

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

6.15.1 Field Documentation

6.15.1.1 unsigned char blue

Blue color intensity 0-0x0F

6.15.1.2 unsigned char green

Green color intensity 0-0x0F

6.15.1.3 unsigned char red

Red color intensity 0-0x0F

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

- IncludeFiles/[CCAuxTypes.h](#)

6.16 LedTimingType Struct Reference

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

Data Fields

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

6.16.1 Field Documentation

6.16.1.1 unsigned char idleTime

LED idle time in 100ms

6.16.1.2 unsigned char nrOfPulses

Pulses per sequences

6.16.1.3 unsigned char offTime

LED off time in 10ms

6.16.1.4 unsigned char onTime

LED on time in 10ms

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

- [IncludeFiles/CCAuxTypes.h](#)

6.17 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

6.17.1 Detailed Description

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

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

Example Usage:

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

```

using namespace std;
using namespace CrossControl;

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

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

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

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

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

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

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

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

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

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

    ls_example(pLightSensor);

    pLightSensor->Release();
}

```

6.17.2 Member Function Documentation

6.17.2.1 virtual eErr getAverageIlluminance (unsigned short * *value*) [pure virtual]

Get average illuminance (light) value from light sensor.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.17.2.2 virtual eErr 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.

Example Usage:

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

6.17.2.3 virtual eErr 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.

6.17.2.4 virtual eErr 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.

6.17.2.5 virtual void Release () [pure virtual]

Delete the [Lightsensor](#) object.

Returns

-

Example Usage:

```

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

ls_example (pLightSensor);

pLightSensor->Release();

```

6.17.2.6 virtual eErr 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.

6.17.2.7 virtual eErr startAverageCalc (unsigned long *averageWndSize*, unsigned long *rejectWndSize*, unsigned long *rejectDeltaInLux*, LightSensorSamplingMode *mode*) [pure virtual]

Start average calculation.

Parameters

<i>averageWndSize</i>	The average window size in nr of samples.
<i>rejectWndSize</i>	The reject window size in nr of samples.
<i>rejectDeltaInLux</i>	The reject delta in lux.
<i>mode</i>	The configured sampling mode.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.17.2.8 virtual eErr stopAverageCalc () [pure virtual]

Stop average calculation.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

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

```

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

- [IncludeFiles/Lightsensor.h](#)

6.18 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

6.18.1 Detailed Description

[Power](#) control access functions

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

Example Usage:

```

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

using namespace std;
using namespace CrossControl;

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

```

```

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

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

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

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

    power_example(pPower);

    pPower->Release();
}

```

6.18.2 Member Function Documentation

6.18.2.1 virtual eErr 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.

6.18.2.2 virtual eErr getBLPowerStatus (CCStatus * *status*) [pure virtual]

Get backlight power status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.18.2.3 virtual eErr 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.

6.18.2.4 virtual eErr 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.

6.18.2.5 virtual eErr 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.

6.18.2.6 virtual eErr getVideoPowerStatus (unsigned char * *videoStatus*) [pure virtual]

Get Video power status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.18.2.7 virtual void Release () [pure virtual]

Delete the [Power](#) object.

Returns

-

Example Usage:

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

power_example (pPower);

pPower->Release();
```

6.18.2.8 virtual eErr setBLPowerStatus (CCStatus status) [pure virtual]

Set backlight power status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.18.2.9 virtual eErr 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.

6.18.2.10 virtual eErr 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.

6.18.2.11 virtual eErr 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:

- IncludeFiles/[Power.h](#)

6.19 PowerMgr Struct Reference

```
#include <PowerMgr.h>
```

Public Member Functions

- virtual eErr registerControlledSuspendOrShutDown (PowerMgrConf conf)=0
- virtual eErr getConfiguration (PowerMgrConf *conf)=0
- virtual eErr getPowerMgrStatus (PowerMgrStatus *status)=0
- virtual eErr setAppReadyForSuspendOrShutdown (void)=0
- virtual eErr hasResumed (bool *resumed)=0
- virtual void Release ()=0

6.19.1 Detailed Description

Access functions for managing application shutdown/cleanup when suspend or shutdown events occur. An application can use this framework to delay a suspend/shutdown until it is ready with any tasks it needs to do.

Example Usage:

```

/* g++ -DLINUX examples/powermgr_example.cpp -lcc-aux -pthread -ldl */
#include <PowerMgr.h>
#include <Battery.h>
#include <CCAuxErrors.h>
#include <assert.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
using namespace CrossControl;
unsigned int suspendConfiguration;
//
// OS-independent Sleep function
//
#ifdef LINUX
static bool OSSleep(unsigned long delay_ms)
{
    struct timespec ts;
    ts.tv_sec = (delay_ms / 1000);
    ts.tv_nsec = (delay_ms - ts.tv_sec * 1000) * 1000000; /* ms */
    if (nanosleep(&ts, NULL) < 0)
    {
        return false;
    }
    return true;
}
#else
#include <windows.h>
static bool OSSleep(DWORD dwMilliseconds)
{
    ::Sleep(dwMilliseconds);
    return true;
}
#endif

//
// Variable used in the console application exit handler. It is set to false when the API handles the
// shutdown.
//
bool os_shutdown = true;

void test_powermgr(PowerMgrHandle pPowerMgr)
{
    if(!pPowerMgr)
        return;

    CrossControl::eErr err;
    PowerMgrConf setConfiguration;

    err = pPowerMgr->getConfiguration(&setConfiguration);
    if(err == ERR_SUCCESS)
    {
        if(setConfiguration == Normal)
        {
            cout << "Normal configuration used... Ctrl-C to exit." << endl;
        }
        else if(setConfiguration == ApplicationControlled)
        {
            cout << "ApplicationControlled configuration used... Ctrl-C to exit." << endl;
        }
        else if(setConfiguration == BatterySuspend)

```

```

    {
        cout << "BatterySuspend configuration used... Ctrl-C to exit." << endl;
    }
}
else
{
    cout << "pPowerMgr->getConfiguration(&setConfiguration) Failed" << endl;
}

while(1)
{
    OSSleep(500);

    PowerMgrStatus status;
    err = pPowerMgr->getPowerMgrStatus(&status);
    if(err == ERR_SUCCESS)
    {
        switch(status)
        {
            case NoRequestsPending: // Wait until a PowerMgr request arrives...
                break;

            case ShutdownPending:
            {
                // Shutdown by means of power button or on/off signal are caught here.
                os_shutdown = false;

                cout << "A shutdown request detected. App should now do what it needs to do before shutdown can
                be performed." << endl;
                cout << "Press Enter when ready to shutdown... " << endl;

                // Make sure to clear cin buffer before read
                std::cin.clear();
                std::cin.ignore(100, '\n');
                cin.get();
                cout << "Signalling that app is ready..." << endl;
                err = pPowerMgr->setAppReadyForSuspendOrShutdown();
                if(err != ERR_SUCCESS)
                {
                    cout << "Error(" << err << ") in function setAppReadyForSuspendOrShutdown: " <<
                    GetErrorStringA(err) << endl;
                }
                return; //exit test app
            }

            case SuspendPending:
            {
                os_shutdown = false;

                cout << "A suspend request detected. App should now do what it needs to do before suspend can be
                performed." << endl;
                cout << "Press Enter when ready to suspend... " << endl;

                // Make sure to clear cin buffer before read
                std::cin.clear();
                std::cin.ignore(100, '\n');
                cin.get();
                cout << "Signalling that app is ready..." << endl;
                err = pPowerMgr->setAppReadyForSuspendOrShutdown();
                if(err != ERR_SUCCESS)
                {
                    cout << "Error(" << err << ") in function setAppReadyForSuspendOrShutdown: " <<
                    GetErrorStringA(err) << endl;
                }
            }
            break;

            default:
                cout << "Error: Invalid status returned from getPowerMgrStatus!" << endl;
                break;
        }
    }

    //Wait for resume after notifying that we are ready to suspend

```

```

if(status == SuspendPending)
{
    bool b = false;
    while(!b)
    {
        OSSleep(100);
        cout << "." << endl;

        err = pPowerMgr->hasResumed(&b);
        if(err != ERR_SUCCESS)
        {
            cout << "Error(" << err << ") in function hasResumed: " <<
GetErrorStringA(err) << endl;
        }
    }
    cout << "System is now resumed from suspend mode!" << endl <<
        "Now we will soon re-register using the registerControlledSuspendOrShutDown function!" << endl;

    // Expecting to get configuration Normal after resume from suspend

    CrossControl::PowerMgrConf conf;
    err = pPowerMgr->getConfiguration(&conf);
    if(err == ERR_SUCCESS)
    {
        switch (conf)
        {
            case Normal:
                cout << "PowerMgrConf is now: Normal" << endl; break;
            case ApplicationControlled:
                cout << "PowerMgrConf is now: ApplicationControlled" << endl; break;
            case BatterySuspend:
                cout << "PowerMgrConf is now: BatterySuspend" << endl; break;
        }
    }
    else
    {
        cout << "Error(" << err << ") in function getConfiguration: " <<
GetErrorStringA(err) << endl;
    }

    // Re-register, do this as soon as possible after resume/startup
    pPowerMgr->registerControlledSuspendOrShutDown(setConfiguration)
;
    if(err == ERR_SUCCESS)
        cout << "Re-registered to powerMgr. Ctrl-C to exit." << endl;
    else
        cout << "Error(" << err << ") in function registerControlledSuspendOrShutDown: " <<
GetErrorStringA(err) << endl;
    }
}
else
{
    cout << "Error(" << err << ") in function getPowerMgrStatus: " <<
GetErrorStringA(err) << endl;
}
}

//
// This function handles OS shutdown initiated from the OS itself and not from power button or on/off
// signal.
//
void fnExit (void)
{
    if(os_shutdown)
    {
        cout << "Test application detected OS shutdown. Wait 5s before exit..." << endl;
        OSSleep(5000);
    }
}

int main(void)
{

```

```

string input = "";
CrossControl::eErr err;
POWERMGRHANDLE pPowerMgr = ::GetPowerMgr();
BATTERYHANDLE pBattery = ::GetBattery();

assert(pPowerMgr);
assert(pBattery);

// Register a separate exit handler for the case where OS is initiating the shutdown. The Application
// must handle this case itself.
atexit(fnExit);

bool bBatt = false;
pBattery->isBatteryPresent(&bBatt);
if(bBatt) // Ask user wih configuration to use...
    cout << "Choose configuration to use, 0 - Normal, 1 - Application Controlled , 2 - Battery Suspend" <<
        endl;
else
    cout << "Choose configuration to use, 0 - Normal, 1 - Application Controlled" << endl;

cin >> suspendConfiguration;
pBattery->Release();

// Register that this application needs to delay suspend/shutdown
// This should be done as soon as possible.
// Then the app must poll getPowerMgrStatus() and allow the suspend/shutdown with
// setAppReadyForSuspendOrShutdown().
// Depending on application design, this might be best handled in a separate thread.
err = pPowerMgr->registerControlledSuspendOrShutDown((
    PowerMgrConf) suspendConfiguration);

if(err == ERR_SUCCESS)
    cout << "Registered to powerMgr." << endl;
else
    cout << "Error(" << err << ") in function registerControlledSuspendOrShutDown: " <<
        GetErrorStringA(err) << endl;

test_powermgr(pPowerMgr);

pPowerMgr->Release();
}

```

6.19.2 Member Function Documentation

6.19.2.1 virtual eErr getConfiguration (PowerMgrConf * conf) [pure virtual]

Get the configuration that is in use.

Parameters

<i>conf</i>	The configuration in use.
-------------	---------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```

CrossControl::PowerMgrConf conf;
err = pPowerMgr->getConfiguration(&conf);
if(err == ERR_SUCCESS)
{
    switch (conf)

```

```

{
    case Normal:
        cout << "PowerMgrConf is now: Normal" << endl; break;
    case ApplicationControlled:
        cout << "PowerMgrConf is now: ApplicationControlled" << endl; break;
    case BatterySuspend:
        cout << "PowerMgrConf is now: BatterySuspend" << endl; break;
    }
}
else
{
    cout << "Error(" << err << ") in function getConfiguration: " <<
    GetErrorStringA(err) << endl;
}
}

```

6.19.2.2 virtual eErr getPowerMgrStatus (PowerMgrStatus * *status*) [pure virtual]

Get the current status of the [PowerMgr](#). This functions should be called periodically, to detect when suspend or shutdown requests arrive.

Parameters

<i>status</i>	The current status.
---------------	---------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```

while(1)
{
    OSSleep(500);

    PowerMgrStatus status;
    err = pPowerMgr->getPowerMgrStatus(&status);
    if(err == ERR_SUCCESS)
    {
        switch(status)
        {
            case NoRequestsPending: // Wait until a PowerMgr request arrives...
                break;

            case ShutdownPending:
            {
                // Shutdown by means of power button or on/off signal are caught here.
                os_shutdown = false;

                cout << "A shutdown request detected. App should now do what it needs to do before shutdown can
                be performed." << endl;
                cout << "Press Enter when ready to shutdown... " << endl;

                // Make sure to clear cin buffer before read
                std::cin.clear();
                std::cin.ignore(100, '\n');
                cin.get();
                cout << "Signalling that app is ready..." << endl;
                err = pPowerMgr->setAppReadyForSuspendOrShutdown();
                if(err != ERR_SUCCESS)

```

```

    {
        cout << "Error(" << err << ") in function setAppReadyForSuspendOrShutdown: " <<
        GetErrorStringA(err) << endl;
    }
    return; //exit test app
}
case SuspendPending:
{
    os_shutdown = false;

    cout << "A suspend request detected. App should now do what it needs to do before suspend can be
    performed." << endl;
    cout << "Press Enter when ready to suspend... " << endl;

    // Make sure to clear cin buffer before read
    std::cin.clear();
    std::cin.ignore(100, '\n');
    cin.get();
    cout << "Signalling that app is ready..." << endl;
    err = pPowerMgr->setAppReadyForSuspendOrShutdown();
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function setAppReadyForSuspendOrShutdown: " <<
        GetErrorStringA(err) << endl;
    }
    break;
default:
    cout << "Error: Invalid status returned from getPowerMgrStatus!" << endl;
    break;
}

//Wait for resume after notifying that we are ready to suspend
if(status == SuspendPending)
{
    bool b = false;
    while(!b)
    {
        OSSleep(100);
        cout << "." << endl;

        err = pPowerMgr->hasResumed(&b);
        if(err != ERR_SUCCESS)
        {
            cout << "Error(" << err << ") in function hasResumed: " <<
            GetErrorStringA(err) << endl;
        }
    }
    cout << "System is now resumed from suspend mode!" << endl <<
        "Now we will soon re-register using the registerControlledSuspendOrShutDown function!" << endl;

    // Expecting to get configuration Normal after resume from suspend

    CrossControl::PowerMgrConf conf;
    err = pPowerMgr->getConfiguration(&conf);
    if(err == ERR_SUCCESS)
    {
        switch (conf)
        {
            case Normal:
                cout << "PowerMgrConf is now: Normal" << endl; break;
            case ApplicationControlled:
                cout << "PowerMgrConf is now: ApplicationControlled" << endl; break;
            case BatterySuspend:
                cout << "PowerMgrConf is now: BatterySuspend" << endl; break;
        }
    }
    else
    {
        cout << "Error(" << err << ") in function getConfiguration: " <<
        GetErrorStringA(err) << endl;
    }
}

```

```

        // Re-register, do this as soon as possible after resume/startup
        pPowerMgr->registerControlledSuspendOrShutdown(setConfiguration)
    ;
    if(err == ERR_SUCCESS)
        cout << "Re-registered to powerMgr. Ctrl-C to exit." << endl;
    else
        cout << "Error(" << err << ") in function registerControlledSuspendOrShutdown: " <<
        GetErrorStringA(err) << endl;
    }
}
else
{
    cout << "Error(" << err << ") in function getPowerMgrStatus: " <<
    GetErrorStringA(err) << endl;
}
}
}

```

6.19.2.3 virtual eErr hasResumed (bool * resumed) [pure virtual]

This function can be used in a suspend-resume scenario. After the application has used [setAppReadyForSuspendOrShutdown\(\)](#) to init the suspend, this function may be polled in order to detect when the system is up and running again. Calling this function before calling [setAppReadyForSuspendOrShutdown](#) will return `resumed = true`.

Returns

error status. 0 = `ERR_SUCCESS`, otherwise error code. See the enum `eErr` for details.

Example Usage:

```

while(1)
{
    OSSleep(500);

    PowerMgrStatus status;
    err = pPowerMgr->getPowerMgrStatus(&status);
    if(err == ERR_SUCCESS)
    {
        switch(status)
        {
            case NoRequestsPending: // Wait until a PowerMgr request arrives...
                break;

            case ShutdownPending:
            {
                // Shutdown by means of power button or on/off signal are caught here.
                os_shutdown = false;

                cout << "A shutdown request detected. App should now do what it needs to do before shutdown can
                be performed." << endl;
                cout << "Press Enter when ready to shutdown... " << endl;

                // Make sure to clear cin buffer before read
                std::cin.clear();
                std::cin.ignore(100, '\n');
                cin.get();
                cout << "Signalling that app is ready..." << endl;
                err = pPowerMgr->setAppReadyForSuspendOrShutdown();
                if(err != ERR_SUCCESS)
                {
                    cout << "Error(" << err << ") in function setAppReadyForSuspendOrShutdown: " <<
                    GetErrorStringA(err) << endl;
                }
            }
        }
    }
}

```

```

    }
    return; //exit test appp
}
case SuspendPending:
{
    os_shutdown = false;

    cout << "A suspend request detected. App should now do what it needs to do before suspend can be
performed." << endl;
    cout << "Press Enter when ready to suspend... " << endl;

    // Make sure to clear cin buffer before read
    std::cin.clear();
    std::cin.ignore(100, '\n');
    cin.get();
    cout << "Signalling that app is ready..." << endl;
    err = pPowerMgr->setAppReadyForSuspendOrShutdown();
    if(err != ERR_SUCCESS)
    {
        cout << "Error(" << err << ") in function setAppReadyForSuspendOrShutdown: " <<
GetErrorStringA(err) << endl;
    }
    break;

default:
    cout << "Error: Invalid status returned from getPowerMgrStatus!" << endl;
    break;
}

//Wait for resume after notifying that we are ready to suspend
if(status == SuspendPending)
{
    bool b = false;
    while(!b)
    {
        OSSleep(100);
        cout << "." << endl;

        err = pPowerMgr->hasResumed(&b);
        if(err != ERR_SUCCESS)
        {
            cout << "Error(" << err << ") in function hasResumed: " <<
GetErrorStringA(err) << endl;
        }
    }
    cout << "System is now resumed from suspend mode!" << endl <<
        "Now we will soon re-register using the registerControlledSuspendOrShutDown function!" << endl;

    // Expecting to get configuration Normal after resume from suspend

    CrossControl::PowerMgrConf conf;
    err = pPowerMgr->getConfiguration(&conf);
    if(err == ERR_SUCCESS)
    {
        switch (conf)
        {
            case Normal:
                cout << "PowerMgrConf is now: Normal" << endl; break;
            case ApplicationControlled:
                cout << "PowerMgrConf is now: ApplicationControlled" << endl; break;
            case BatterySuspend:
                cout << "PowerMgrConf is now: BatterySuspend" << endl; break;
        }
    }
    else
    {
        cout << "Error(" << err << ") in function getConfiguration: " <<
GetErrorStringA(err) << endl;
    }

    // Re-register, do this as soon as possible after resume/startup
    pPowerMgr->registerControlledSuspendOrShutDown(setConfiguration)

```

```

;
    if(err == ERR_SUCCESS)
        cout << "Re-registered to powerMgr. Ctrl-C to exit." << endl;
    else
        cout << "Error(" << err << ") in function registerControlledSuspendOrShutDown: " <<
        GetErrorStringA(err) << endl;
    }
}
else
{
    cout << "Error(" << err << ") in function getPowerMgrStatus: " <<
    GetErrorStringA(err) << endl;
}
}
}

```

6.19.2.4 virtual eErr registerControlledSuspendOrShutDown (PowerMgrConf conf) [pure virtual]

Configure the [PowerMgr](#). Call this function once initially to turn on the functionality.

Parameters

<i>conf</i>	The configuration to use.
-------------	---------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```

CrossControl::eErr err;
POWERMGRHANDLE pPowerMgr = ::GetPowerMgr();
BATTERYHANDLE pBattery = ::GetBattery();

assert(pPowerMgr);
assert(pBattery);

// Register a separate exit handler for the case where OS is initiating the shutdown. The Application
// must handle this case itself.
atexit(fnExit);

bool bBatt = false;
pBattery->isBatteryPresent(&bBatt);
if(bBatt) // Ask user wich configuration to use...
    cout << "Choose configuration to use, 0 - Normal, 1 - Application Controlled , 2 - Battery Suspend" <<
    endl;
else
    cout << "Choose configuration to use, 0 - Normal, 1 - Application Controlled" << endl;

cin >> suspendConfiguration;
pBattery->Release();

// Register that this application needs to delay suspend/shutdown
// This should be done as soon as possible.
// Then the app must poll getPowerMgrStatus() and allow the suspend/shutdown with
// setAppReadyForSuspendOrShutdown().
// Depending on application design, this might be best handled in a separate thread.
err = pPowerMgr->registerControlledSuspendOrShutDown((
    PowerMgrConf) suspendConfiguration);

if(err == ERR_SUCCESS)
    cout << "Registered to powerMgr." << endl;

```

```

else
    cout << "Error(" << err << ") in function registerControlledSuspendOrShutDown: " <<
        GetErrorStringA(err) << endl;

test_powermgr (pPowerMgr);

pPowerMgr->Release();

```

6.19.2.5 virtual void Release () [pure virtual]

Delete the [PowerMgr](#) object.

Returns

-

Example Usage:

```

CrossControl::eErr err;
POWERMGRHANDLE pPowerMgr = ::GetPowerMgr();
BATTERYHANDLE pBattery = ::GetBattery();

assert (pPowerMgr);
assert (pBattery);

// Register a separate exit handler for the case where OS is initiating the shutdown. The Application
// must handle this case itself.
atexit (fnExit);

bool bBatt = false;
pBattery->isBatteryPresent (&bBatt);
if (bBatt) // Ask user wich configuration to use...
    cout << "Choose configuration to use, 0 - Normal, 1 - Application Controlled , 2 - Battery Suspend" <<
        endl;
else
    cout << "Choose configuration to use, 0 - Normal, 1 - Application Controlled" << endl;

cin >> suspendConfiguration;
pBattery->Release();

// Register that this application needs to delay suspend/shutdown
// This should be done as soon as possible.
// Then the app must poll getPowerMgrStatus() and allow the suspend/shutdown with
// setAppReadyForSuspendOrShutdown().
// Depending on application design, this might be best handled in a separate thread.
err = pPowerMgr->registerControlledSuspendOrShutDown((
    PowerMgrConf) suspendConfiguration);

if (err == ERR_SUCCESS)
    cout << "Registered to powerMgr." << endl;
else
    cout << "Error(" << err << ") in function registerControlledSuspendOrShutDown: " <<
        GetErrorStringA(err) << endl;

test_powermgr (pPowerMgr);

pPowerMgr->Release();

```

6.19.2.6 virtual eErr setAppReadyForSuspendOrShutdown (void) [pure virtual]

Acknowledge that the application is ready for suspend/shutdown. Should be called after a request has been received in order to execute the request. The application must

acknowledge a request within 20s from when it arrives.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
while(1)
{
    OSSleep(500);

    PowerMgrStatus status;
    err = pPowerMgr->getPowerMgrStatus(&status);
    if(err == ERR_SUCCESS)
    {
        switch(status)
        {
            case NoRequestsPending: // Wait until a PowerMgr request arrives...
                break;

            case ShutdownPending:
            {
                // Shutdown by means of power button or on/off signal are caught here.
                os_shutdown = false;

                cout << "A shutdown request detected. App should now do what it needs to do before shutdown can
                be performed." << endl;
                cout << "Press Enter when ready to shutdown... " << endl;

                // Make sure to clear cin buffer before read
                std::cin.clear();
                std::cin.ignore(100, '\n');
                cin.get();
                cout << "Signalling that app is ready..." << endl;
                err = pPowerMgr->setAppReadyForSuspendOrShutdown();
                if(err != ERR_SUCCESS)
                {
                    cout << "Error(" << err << ") in function setAppReadyForSuspendOrShutdown: " <<
                    GetErrorStringA(err) << endl;
                }
                return; //exit test app
            }
            case SuspendPending:
            {
                os_shutdown = false;

                cout << "A suspend request detected. App should now do what it needs to do before suspend can be
                performed." << endl;
                cout << "Press Enter when ready to suspend... " << endl;

                // Make sure to clear cin buffer before read
                std::cin.clear();
                std::cin.ignore(100, '\n');
                cin.get();
                cout << "Signalling that app is ready..." << endl;
                err = pPowerMgr->setAppReadyForSuspendOrShutdown();
                if(err != ERR_SUCCESS)
                {
                    cout << "Error(" << err << ") in function setAppReadyForSuspendOrShutdown: " <<
                    GetErrorStringA(err) << endl;
                }
            }
            break;
        }
        default:
            cout << "Error: Invalid status returned from getPowerMgrStatus!" << endl;
            break;
    }
}
```

```

    }

    //Wait for resume after notifying that we are ready to suspend
    if(status == SuspendPending)
    {
        bool b = false;
        while(!b)
        {
            OSSleep(100);
            cout << "." << endl;

            err = pPowerMgr->hasResumed(&b);
            if(err != ERR_SUCCESS)
            {
                cout << "Error(" << err << ") in function hasResumed: " <<
                GetErrorStringA(err) << endl;
            }
        }
        cout << "System is now resumed from suspend mode!" << endl <<
        "Now we will soon re-register using the registerControlledSuspendOrShutDown function!" << endl;

        // Expecting to get configuration Normal after resume from suspend

        CrossControl::PowerMgrConf conf;
        err = pPowerMgr->getConfiguration(&conf);
        if(err == ERR_SUCCESS)
        {
            switch (conf)
            {
                case Normal:
                    cout << "PowerMgrConf is now: Normal" << endl; break;
                case ApplicationControlled:
                    cout << "PowerMgrConf is now: ApplicationControlled" << endl; break;
                case BatterySuspend:
                    cout << "PowerMgrConf is now: BatterySuspend" << endl; break;
            }
        }
        else
        {
            cout << "Error(" << err << ") in function getConfiguration: " <<
            GetErrorStringA(err) << endl;
        }

        // Re-register, do this as soon as possible after resume/startup
        pPowerMgr->registerControlledSuspendOrShutDown(setConfiguration)
    ;
    if(err == ERR_SUCCESS)
        cout << "Re-registered to powerMgr. Ctrl-C to exit." << endl;
    else
        cout << "Error(" << err << ") in function registerControlledSuspendOrShutDown: " <<
        GetErrorStringA(err) << endl;
    }
}
else
{
    cout << "Error(" << err << ") in function getPowerMgrStatus: " <<
    GetErrorStringA(err) << endl;
}
}
}

```

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

- IncludeFiles/[PowerMgr.h](#)

6.20 received_video Struct Reference

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

Data Fields

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

6.20.1 Field Documentation

6.20.1.1 unsigned char [received_framerate](#)

6.20.1.2 unsigned short [received_height](#)

6.20.1.3 unsigned short [received_width](#)

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

- [IncludeFiles/CCAuxTypes.h](#)

6.21 Smart Struct Reference

```
#include <Smart.h>
```

Public Member Functions

- virtual [eErr getRemainingLifeTime](#) (unsigned char *lifetimepercent)=0
- virtual [eErr getDeviceSerial](#) (char *buff, int len)=0
- virtual [eErr getInitialTime](#) (time_t *time)=0
- virtual void [Release](#) ()=0

6.21.1 Detailed Description

Get S.M.A.R.T. data and information from the secondary storage device (CF/SD/HDD) where the OS is installed.

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

Example Usage:

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

using namespace std;

void show_card_data(SMARTHANDLE pSmart)
{
```

```

if(!pSmart)
    return;

CrossControl::eErr err;

cout << "Smart class example:" << endl;

char serial[21];
err = pSmart->getDeviceSerial (serial, sizeof(serial));
if (CrossControl::ERR_SUCCESS == err)
{
    cout << "Device serial number: " << serial << endl;
}
else
{
    cout << "Error(" << err << ") in function getDeviceSerial: " <<
        GetErrorStringA(err) << endl;
}

unsigned char life;
err = pSmart->getRemainingLifeTime (&life);
if (CrossControl::ERR_SUCCESS == err)
{
    cout << "Estimated remaining lifetime: " << (int)life << "%" << endl;
}
else
{
    cout << "Error(" << err << ") in function getRemainingLifeTime: " <<
        GetErrorStringA(err) << endl;
}

time_t initialTime;
struct tm * timeinfo;
err = pSmart->getInitialTime (&initialTime);
if (CrossControl::ERR_SUCCESS == err)
{
    cout << "Device was initially timestamped on: ";
    timeinfo = localtime (&initialTime);
    cout << asctime(timeinfo) << endl;
}
else
{
    cout << "Error(" << err << ") in function getInitialTime: " <<
        GetErrorStringA(err) << endl;
}
}

int main(void)
{
    SMARTHANDLE pSmart = ::GetSmart();
    assert(pSmart);

    show_card_data(pSmart);

    pSmart->Release();
}

```

6.21.2 Member Function Documentation

6.21.2.1 virtual eErr getDeviceSerial (char * buff, int len) [pure virtual]

Get serial number of the secondary storage device.

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. At least an 21 bytes buffer size must be used since the serial number can be 20 bytes + trailing zero.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
char serial[21];
err = pSmart->getDeviceSerial (serial, sizeof(serial));
if (CrossControl::ERR_SUCCESS == err)
{
    cout << "Device serial number: " << serial << endl;
}
else
{
    cout << "Error(" << err << ") in function getDeviceSerial: " <<
        GetErrorStringA(err) << endl;
}
```

6.21.2.2 virtual eErr getInitialTime (time_t * time) [pure virtual]

Get the date/time when the SMART monitoring began for this storage device. This time is either when the card first was used or when the system software was updated to support S.M.A.R.T. monitoring for the first time. Logging of time is based on the local time of the computer at the time of logging and may therefore not always be accurate.

Parameters

<i>time</i>	A 32bit time_t value representing the number of seconds elapsed since 00:00 hours, Jan 1, 1970 UTC.
-------------	---

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
time_t initialTime;
struct tm * timeinfo;
err = pSmart->getInitialTime (&initialTime);
if (CrossControl::ERR_SUCCESS == err)
{
    cout << "Device was initially timestamped on: ";
    timeinfo = localtime (&initialTime);
    cout << asctime(timeinfo) << endl;
}
else
{
    cout << "Error(" << err << ") in function getInitialTime: " <<
        GetErrorStringA(err) << endl;
}
```

6.21.2.3 virtual eErr getRemainingLifeTime (unsigned char * *lifetimepercent*) [pure virtual]

Get remaining lifetime of the secondary storage device.

Parameters

<i>lifetimepercent</i>	The expected remaining lifetime (0..100%).
------------------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char life;
err = pSmart->getRemainingLifeTime (&life);
if (CrossControl::ERR_SUCCESS == err)
{
    cout << "Estimated remaining lifetime: " << (int)life << "%" << endl;
}
else
{
    cout << "Error(" << err << ") in function getRemainingLifeTime: " <<
        GetErrorStringA(err) << endl;
}
```

6.21.2.4 virtual void Release () [pure virtual]

Delete the [Smart](#) object.

Returns

-

Example Usage:

```
SMARTHANDLE pSmart = ::GetSmart();
assert(pSmart);

show_card_data(pSmart);

pSmart->Release();
```

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

- IncludeFiles/[Smart.h](#)

6.22 Telematics Struct Reference

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

Public Member Functions

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

6.22.1 Detailed Description

[Power](#) control and status functions for the optional telematics add-on card

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

Example Usage:

```

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

using namespace std;
using namespace CrossControl;

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

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

```

```

        GetErrorStringA(err) << endl;
    return;
}

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

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

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

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

err = pTelematics->getGPSPowerStatus(&status);
if(err == ERR_SUCCESS)
{
    cout << "GPS power is " << ((status == Enabled)? "ON" : "OFF") << endl;
}
else if(err == ERR_TELEMATICS_GPS_NOT_AVAILABLE)

```

```

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

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

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

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

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

```

```
int main(void)
{
    TELEMATICSHANDLE pTelematics = ::GetTelematics();
    assert(pTelematics);

    telematics_example(pTelematics);

    pTelematics->Release();
}
```

6.22.2 Member Function Documentation

6.22.2.1 virtual eErr getBTPowerStatus (CCStatus * status) [pure virtual]

Get Bluetooth power status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.2 virtual eErr getBTStartupPowerStatus (CCStatus * status) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

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

```

6.22.2.3 virtual eErr getGPRSPowerStatus (CCStatus * *status*) [pure virtual]

Get GPRS power status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```

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

```

6.22.2.4 virtual eErr getGPRSStartUpPowerStatus (CCStatus * *status*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.5 virtual eErr getGPSAntennaStatus (CCStatus * status) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.6 virtual eErr getGPSPowerStatus (CCStatus * status) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.7 virtual eErr getGPSStartupPowerStatus (CCStatus * *status*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.8 virtual eErr getTelematicsAvailable (CCStatus * *status*) [pure virtual]

Is a telematics add-on card installed?

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.9 virtual eErr getWLANPowerStatus (CCStatus * *status*) [pure virtual]

Get WLAN power status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.10 virtual eErr getWLANStartUpPowerStatus (CCStatus * *status*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.11 virtual void Release () [pure virtual]

Delete the [Telematics](#) object.

Returns

-

Example Usage:

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

telematics_example(pTelematics);

pTelematics->Release();
```

6.22.2.12 virtual eErr setBTPowerStatus (CCStatus *status*) [pure virtual]

Set Bluetooth power status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.13 `virtual eErr setBTStartupPowerStatus (CCStatus status)` [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.14 `virtual eErr 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.

6.22.2.15 `virtual eErr 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.

6.22.2.16 virtual eErr setGPSPowerStatus (CCStatus *status*) [pure virtual]

Set GPS power status.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.17 virtual eErr setGPSStartUpPowerStatus (CCStatus *status*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.18 virtual eErr 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.

6.22.2.19 virtual eErr 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:

- IncludeFiles/[Telematics.h](#)

6.23 TimerType Struct Reference

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

Data Fields

- unsigned long [TotRunTime](#)
- unsigned long [TotSuspTime](#)
- unsigned long [TotHeatTime](#)
- unsigned long [RunTime40_60](#)
- unsigned long [RunTime60_70](#)
- unsigned long [RunTime70_80](#)
- unsigned long [Above80RunTime](#)

6.23.1 Detailed Description

[Diagnostic](#) timer data

6.23.2 Field Documentation

6.23.2.1 unsigned long Above80RunTime

Total runtime in 70-80deg (minutes)

6.23.2.2 unsigned long RunTime40_60

Total heating time (minutes)

6.23.2.3 unsigned long RunTime60_70

Total runtime in 40-60deg (minutes)

6.23.2.4 unsigned long RunTime70_80

Total runtime in 60-70deg (minutes)

6.23.2.5 unsigned long TotHeatTime

Total suspend time (minutes)

6.23.2.6 unsigned long TotRunTime

6.23.2.7 unsigned long TotSuspTime

Total running time (minutes)

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

- [IncludeFiles/CCAuxTypes.h](#)

6.24 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

6.24.1 Detailed Description

Touch Screen settings

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

Example Usage:

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

using namespace std;
using namespace CrossControl;

void touchscreen_example(TOUCHSCREENHANDLE pTouchScreen)
{
```

```

CrossControl::eErr err;
unsigned short rightclicktime, debouncetime;
TouchScreenModeSettings ts_mode;

err = pTouchScreen->getMode(&ts_mode);
if(err == ERR_SUCCESS)
{
    switch(ts_mode)
    {
        case MOUSE_NEXT_BOOT: cout << "USB profile is set to Mouse profile (active next boot)" <
        < endl; break;
        case TOUCH_NEXT_BOOT: cout << "USB profile is set to Touch profile (active next boot)" <
        < endl; break;
        case MOUSE_NOW: cout << "USB profile is set to Mouse profile" << endl; break;
        case TOUCH_NOW: cout << "USB profile is set to Touch profile" << endl; break;
        default: cout << "Error: invalid setting returned from getMode" << endl; break;
    }
}
else
{
    cout << "Error(" << err << ") in function getMode: " << GetErrorStringA(err) << endl;
}

err = pTouchScreen->getMouseRightClickTime(&rightclicktime);
if(err == ERR_SUCCESS)
{
    cout << "Right click time is set to: " << (int)rightclicktime << " ms" << endl;
}
else
{
    cout << "Error(" << err << ") in function getMouseRightClickTime: " <<
    GetErrorStringA(err) << endl;
}

err = pTouchScreen->getAdvancedSetting(TS_DEBOUNCE_TIME, &debouncetime)
;
if(err == ERR_SUCCESS)
{
    cout << "Touchscreen debounce time is set to: " << (int)debouncetime << " ms" << endl;
}
else
{
    cout << "Error(" << err << ") in function getAdvancedSetting: " <<
    GetErrorStringA(err) << endl;
}
}

int main(void)
{
    TOUCHSCREENHANDLE pTouchScreen = ::GetTouchScreen();
    assert(pTouchScreen);

    touchscreen_example(pTouchScreen);

    pTouchScreen->Release();
}

```

6.24.2 Member Function Documentation

6.24.2.1 virtual eErr getAdvancedSetting (TSAdvancedSettingsParameter *param*, unsigned short * *data*) [pure virtual]

Get advanced touch screen settings. See the description of TSAdvancedSettingsParameter for a description of the parameters.

Parameters

<i>param</i>	The setting to get.
<i>data</i>	The current data for the setting.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTouchScreen->getAdvancedSetting(TS_DEBOUNCE_TIME, &debouncetime)
;
if(err == ERR_SUCCESS)
{
    cout << "Touchscreen debounce time is set to: " << (int)debouncetime << " ms" << endl;
}
else
{
    cout << "Error(" << err << ") in function getAdvancedSetting: " <<
        GetErrorStringA(err) << endl;
}
```

6.24.2.2 virtual eErr getMode (TouchScreenModeSettings * *config*) [pure virtual]

Get Touch Screen mode. Gets the current mode of the USB profile.

Parameters

<i>config</i>	The current mode.
---------------	-------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTouchScreen->getMode(&ts_mode);
if(err == ERR_SUCCESS)
{
    switch(ts_mode)
    {
        case MOUSE_NEXT_BOOT: cout << "USB profile is set to Mouse profile (active next boot)" <
            < endl; break;
        case TOUCH_NEXT_BOOT: cout << "USB profile is set to Touch profile (active next boot)" <
            < endl; break;
        case MOUSE_NOW: cout << "USB profile is set to Mouse profile" << endl; break;
        case TOUCH_NOW: cout << "USB profile is set to Touch profile" << endl; break;
        default: cout << "Error: invalid setting returned from getMode" << endl; break;
    }
}
else
{
    cout << "Error(" << err << ") in function getMode: " << GetErrorStringA(err) << endl;
}
```

6.24.2.3 `virtual eErr getMouseRightClickTime (unsigned short * time)` [pure virtual]

Get mouse right click time. Applies only to the mouse profile. Use the OS settings for the touch profile.

Parameters

<i>time</i>	The right click time, in milliseconds.
-------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
err = pTouchScreen->getMouseRightClickTime(&rightclicktime);
if(err == ERR_SUCCESS)
{
    cout << "Right click time is set to: " << (int)rightclicktime << " ms" << endl;
}
else
{
    cout << "Error(" << err << ") in function getMouseRightClickTime: " <<
        GetErrorStringA(err) << endl;
}
```

6.24.2.4 `virtual void Release ()` [pure virtual]

Delete the [TouchScreen](#) object.

Returns

-

Example Usage:

```
TOUCHSCREENHANDLE pTouchScreen = ::GetTouchScreen();
assert(pTouchScreen);

touchscreen_example(pTouchScreen);

pTouchScreen->Release();
```

6.24.2.5 `virtual eErr setAdvancedSetting (TSAdvancedSettingsParameter param, unsigned short data)` [pure virtual]

Set advanced touch screen settings. See the description of [TSAdvancedSettingsParameter](#) for a description of the parameters.

Parameters

<i>param</i>	The setting to set.
<i>data</i>	The data value to set.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.24.2.6 virtual eErr 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.

6.24.2.7 virtual eErr 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:

- IncludeFiles/[TouchScreen.h](#)

6.25 TouchScreenCalib Struct Reference

```
#include <TouchScreenCalib.h>
```

Public Member Functions

- virtual `eErr SetMode (CalibrationModeSettings mode)=0`
- virtual `eErr GetMode (CalibrationModeSettings *mode)=0`
- virtual `eErr SetCalibrationPoint (unsigned char pointNr)=0`
- virtual `eErr CheckCalibrationPointFinished (bool *finished, unsigned char pointNr)=0`
- virtual `eErr GetConfigParam (CalibrationConfigParam param, unsigned short *value)=0`
- virtual `eErr SetConfigParam (CalibrationConfigParam param, unsigned short value)=0`
- virtual void `Release ()=0`

6.25.1 Detailed Description

Touch Screen Calibration interface

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

6.25.2 Member Function Documentation

6.25.2.1 `virtual eErr CheckCalibrationPointFinished (bool * finished, unsigned char pointNr)`
`[pure virtual]`

Check if a calibration point is finished

Parameters

<i>finished</i>	Is current point finished?
<i>pointNr</i>	Calibration point number (1 to total number of points)

Returns

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

6.25.2.2 `virtual eErr GetConfigParam (CalibrationConfigParam param, unsigned short * value)`
`[pure virtual]`

Get calibration config parameters

Parameters

<i>param</i>	<code>Config</code> parameter
<i>value</i>	Parameter value

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.25.2.3 `virtual eErr GetMode (CalibrationModeSettings * mode)` [pure virtual]

Get mode of front controller.

Parameters

<i>mode</i>	Current calibration mode
-------------	--------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.25.2.4 `virtual void Release ()` [pure virtual]

Delete the [TouchScreenCalib](#) object.

Returns

-

6.25.2.5 `virtual eErr SetCalibrationPoint (unsigned char pointNr)` [pure virtual]

Set calibration point

Parameters

<i>pointNr</i>	Calibration point number (1 to total number of points)
----------------	--

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.25.2.6 `virtual eErr SetConfigParam (CalibrationConfigParam param, unsigned short value)` [pure virtual]

Set calibration config parameters

Parameters

<i>param</i>	Config parameter
<i>value</i>	parameter value

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.25.2.7 virtual eErr SetMode (CalibrationModeSettings *mode*) [pure virtual]

Set mode of front controller.

Parameters

<i>mode</i>	Selected calibration mode
-------------	---------------------------

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

- IncludeFiles/[TouchScreenCalib.h](#)

6.26 UpgradeStatus Struct Reference

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

Data Fields

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

6.26.1 Detailed Description

Upgrade Status

6.26.2 Field Documentation

6.26.2.1 enum [UpgradeAction](#) *currentAction*

6.26.2.2 eErr *errorCode*

Represents the percentage of completion of the current action

6.26.2.3 unsigned char percent

The current action.

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

- [IncludeFiles/CCAuxTypes.h](#)

6.27 version_info Struct Reference

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

Data Fields

- unsigned char [major](#)
- unsigned char [minor](#)
- unsigned char [release](#)
- unsigned char [build](#)

6.27.1 Field Documentation

6.27.1.1 unsigned char build

version build number

6.27.1.2 unsigned char major

version major number

6.27.1.3 unsigned char minor

version minor number

6.27.1.4 unsigned char release

version release number

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

- [IncludeFiles/CCAuxTypes.h](#)

6.28 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

6.28.1 Detailed Description

Analog [Video](#)

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

6.28.2 Member Function Documentation

6.28.2.1 virtual eErr 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.

6.28.2.2 virtual eErr 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.

6.28.2.3 virtual eErr freeBmpBuffer (char * *bmpBuffer*) [pure virtual]

Free the memory allocated for BMP buffer.

Parameters

<i>bmpBuffer</i>	The bmp buffer to free.
------------------	-------------------------

Returns

error status.

6.28.2.4 virtual eErr 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.

6.28.2.5 virtual eErr 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.

6.28.2.6 virtual eErr 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.

6.28.2.7 `virtual eErr 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.

6.28.2.8 `virtual eErr 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.

6.28.2.9 `virtual eErr 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.

6.28.2.10 virtual eErr 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.

6.28.2.11 virtual eErr 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.

6.28.2.12 virtual eErr 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.

6.28.2.13 `virtual eErr 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.

6.28.2.14 `virtual eErr 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.

6.28.2.15 `virtual eErr 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.

6.28.2.16 `virtual eErr minimize () [pure virtual]`

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.17 `virtual void Release () [pure virtual]`

Delete the [Video](#) object.

Returns

-

6.28.2.18 `virtual eErr 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.

6.28.2.19 virtual eErr 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.

6.28.2.20 virtual eErr 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.

6.28.2.21 virtual eErr 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.

6.28.2.22 `virtual eErr 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.

6.28.2.23 `virtual eErr 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.

6.28.2.24 `virtual eErr 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.

6.28.2.25 virtual eErr 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.

6.28.2.26 virtual eErr 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.

6.28.2.27 virtual eErr 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.

6.28.2.28 virtual eErr 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.

6.28.2.29 virtual eErr 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.

6.28.2.30 virtual eErr 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 + line count * row count * 4. The size of the raw image is when interlaced = true 0x100 + line count * 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:

- IncludeFiles/Video.h

6.29 video_dec_command Struct Reference

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

Data Fields

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

6.29.1 Field Documentation

6.29.1.1 unsigned char [decoder_register](#)

6.29.1.2 unsigned char [register_value](#)

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

- [IncludeFiles/CCAuxTypes.h](#)

Chapter 7

File Documentation

7.1 IncludeFiles/About.h File Reference

Data Structures

- struct [About](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::About](#) * [ABOUTHANDLE](#)

Functions

- [EXTERN_C](#) [CCAUXDLL_API](#)
[ABOUTHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetAbout](#) (void)

7.1.1 Typedef Documentation

7.1.1.1 typedef [CrossControl::About](#)* [ABOUTHANDLE](#)

7.1.2 Function Documentation

7.1.2.1 [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();
```

7.2 IncludeFiles/Adc.h File Reference

Data Structures

- struct [Adc](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Adc](#) * ADCHANDLE

Enumerations

- enum [VoltageEnum](#) {
VOLTAGE_24VIN = 0, VOLTAGE_24V, VOLTAGE_12V, VOLTAGE_12-VID,
VOLTAGE_5V, VOLTAGE_3V3, VOLTAGE_VTFT, VOLTAGE_5VSTB,
VOLTAGE_1V9, VOLTAGE_1V8, VOLTAGE_1V5, VOLTAGE_1V2,
VOLTAGE_1V05, VOLTAGE_1V0, VOLTAGE_0V9, VOLTAGE_VREF_I-NT }

Functions

- EXTERN_C CCAUXDLL_API
[ADCHANDLE](#)
CCAUXDLL_CALLING_CONV [GetAdc](#) (void)

7.2.1 Typedef Documentation

7.2.1.1 typedef `CrossControl::Adc*` `ADCHANDLE`

7.2.2 Function Documentation

7.2.2.1 `EXTERN_C CCAUXDLL_API ADCHANDLE CCAUXDLL_CALLING_CONV GetAdc (void)`

Factory function that creates instances of the `Adc` object.

Returns

`ABOUTHANDLE` to an allocated `Adc` structure. The returned handle needs to be deallocated using the `Adc::Release()` method when it's no longer needed. Returns `NULL` if it fails to allocate memory.

Example Usage:

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

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

pAdc->Release();
```

7.3 IncludeFiles/AuxVersion.h File Reference

Data Structures

- struct [AuxVersion](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef `CrossControl::AuxVersion *` `AUXVERSIONHANDLE`

Functions

- `EXTERN_C CCAUXDLL_API AUXVERSIONHANDLE CCAUXDLL_CALLING_CONV GetAuxVersion (void)`

7.3.1 Typedef Documentation

7.3.1.1 `typedef CrossControl::AuxVersion* AUXVERSIONHANDLE`

7.3.2 Function Documentation

7.3.2.1 `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();
```

7.4 IncludeFiles/Backlight.h File Reference

Data Structures

- struct `Backlight`

Namespaces

- namespace `CrossControl`

Typedefs

- typedef `CrossControl::Backlight * BACKLIGHTHANDLE`

Functions

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

7.4.1 Typedef Documentation

- 7.4.1.1 typedef `CrossControl::Backlight*` BACKLIGHTHANDLE

7.4.2 Function Documentation

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

Factory function that creates instances of the Backlight object.

Returns

BACKLIGHTHANDLE to an allocated Backlight structure. The returned handle needs to be deallocated using the `Backlight::Release()` method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

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

change_backlight (pBacklight);

pBacklight->Release();
```

7.5 IncludeFiles/Battery.h File Reference

Data Structures

- struct [BatteryTimerType](#)
- struct [Battery](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef `CrossControl::Battery *` BATTERYHANDLE

Enumerations

- enum `ChargingStatus` {
`ChargingStatus_NoCharge` = 0, `ChargingStatus_Charging` = 1, `ChargingStatus_FullyCharged` = 2, `ChargingStatus_TempLow` = 3,
`ChargingStatus_TempHigh` = 4, `ChargingStatus_Unknown` = 5 }
- enum `PowerSource` { `PowerSource_Battery` = 0, `PowerSource_ExternalPower` = 1 }
- enum `ErrorStatus` {
`ErrorStatus_NoError` = 0, `ErrorStatus_ThermistorTempSensor` = 1, `ErrorStatus_SecondaryTempSensor` = 2, `ErrorStatus_ChargeFail` = 3,
`ErrorStatus_Overcurrent` = 4, `ErrorStatus_Init` = 5 }

Functions

- `EXTERN_C CCAUXDLL_API BATTERYHANDLE CCAUXDLL_CALLING_CONV GetBattery (void)`

7.5.1 Typedef Documentation

- 7.5.1.1 `typedef CrossControl::Battery* BATTERYHANDLE`

7.5.2 Function Documentation

- 7.5.2.1 `EXTERN_C CCAUXDLL_API BATTERYHANDLE CCAUXDLL_CALLING_CONV GetBattery (void)`

Factory function that creates instances of the Battery object.

Returns

`BATTERYHANDLE` to an allocated battery structure. The returned handle needs to be deallocated using the `Battery::Release()` method when it's no longer needed. Returns `NULL` if it fails to allocate memory.

Example Usage:

```
BATTERYHANDLE pBattery = ::GetBattery();
assert(pBattery);

readBatteryInfo(pBattery);

pBattery->Release();
```

7.6 IncludeFiles/Buzzer.h File Reference

Data Structures

- struct [Buzzer](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Buzzer](#) * [BUZZERHANDLE](#)

Functions

- [EXTERN_C](#) [CCAUXDLL_API](#)
[BUZZERHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetBuzzer](#) (void)

7.6.1 Typedef Documentation

7.6.1.1 typedef [CrossControl::Buzzer](#)* [BUZZERHANDLE](#)

7.6.2 Function Documentation

7.6.2.1 [EXTERN_C](#) [CCAUXDLL_API](#) [BUZZERHANDLE](#) [CCAUXDLL_CALLING_CONV](#) [GetBuzzer](#) (void)

Factory function that creates instances of the Buzzer object.

Returns

[BUZZERHANDLE](#) to an allocated Buzzer structure. The returned handle needs to be deallocated using the [Buzzer::Release\(\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
BUZZERHANDLE pBuzzer = ::GetBuzzer ();  
assert (pBuzzer);  
  
play_beeps (pBuzzer);  
  
pBuzzer->Release ();
```

7.7 IncludeFiles/CanSetting.h File Reference

Data Structures

- struct [CanSetting](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::CanSetting](#) * [CANSETTINGHANDLE](#)

Functions

- [EXTERN_C](#) [CCAUXDLL_API](#)
[CANSETTINGHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetCanSetting](#) (void)

7.7.1 Typedef Documentation

7.7.1.1 typedef [CrossControl::CanSetting](#)* [CANSETTINGHANDLE](#)

7.7.2 Function Documentation

7.7.2.1 [EXTERN_C](#) [CCAUXDLL_API](#) [CANSETTINGHANDLE](#) [CCAUXDLL_CALLING_CONV](#) [GetCanSetting](#) (void)

Factory function that creates instances of the [CanSetting](#) object.

Returns

[CANSETTINGHANDLE](#) to an allocated [CanSetting](#) structure. The returned handle needs to be deallocated using the [CanSetting::Release\(\)](#) method when it's no longer needed. Returns [NULL](#) if it fails to allocate memory.

Example Usage:

```
CANSETTINGHANDLE pCanSetting = ::GetCanSetting();  
assert (pCanSetting);  
  
read_cansettings (pCanSetting);  
  
pCanSetting->Release();
```

7.8 IncludeFiles/CCAuxErrors.h File Reference

Namespaces

- namespace [CrossControl](#)

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)

7.9 IncludeFiles/CCAuxTypes.h File Reference

Data Structures

- struct [received_video](#)
- struct [video_dec_command](#)
- struct [version_info](#)
- struct [BuzzerSetup](#)
- struct [LedTimingType](#)
- struct [LedColorMixType](#)
- struct [TimerType](#)
- struct [UpgradeStatus](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef struct version_info [VersionType](#)

Enumerations

- 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, ER-
R_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, ER-
R_NEWER_FRONT_VERSION_REQUIRED = 27,
ERR_TELEMATICS_GPRS_NOT_AVAILABLE = 28, ERR_TELEMATICS-
_WLAN_NOT_AVAILABLE = 29, ERR_TELEMATICS_BT_NOT_AVAIL-
ABLE = 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, ST-
D_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, TEM-
P_OTHER = 7 }
• enum UpgradeAction {
UPGRADE_INIT, UPGRADE_PREP_COM, UPGRADE_READING_FILE, U-
PGRADE_CONVERTING_FILE,
UPGRADE_FLASHING, UPGRADE_VERIFYING, UPGRADE_COMPLET-
E, UPGRADE_COMPLETE_WITH_ERRORS }
• enum CCAuxColor {
RED = 0, GREEN, BLUE, CYAN,
MAGENTA, YELLOW, UNDEFINED_COLOR }

```

7.10 IncludeFiles/CCPlatform.h File Reference

7.11 IncludeFiles/Config.h File Reference

Data Structures

- struct [Config](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Config](#) * [CONFIGHANDLE](#)

Functions

- [EXTERN_C](#) [CCAUXDLL_API](#)
[CONFIGHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetConfig](#) (void)

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)

7.11.1 Typedef Documentation

7.11.1.1 typedef [CrossControl::Config](#)* [CONFIGHANDLE](#)

7.11.2 Function Documentation

7.11.2.1 [EXTERN_C](#) [CCAUXDLL_API](#) [CONFIGHANDLE](#) [CCAUXDLL_CALLING_CONV](#)
[GetConfig](#) (void)

Factory function that creates instances of the [Config](#) object.

Returns

[CONFIGHANDLE](#) to an allocated [Config](#) structure. The returned handle needs to be deallocated using the [Config::Release\(\)](#) method when it's no longer needed. Returns [NULL](#) if it fails to allocate memory.

Example Usage:

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

conf_example(pConfig);

pConfig->Release();
```

7.12 IncludeFiles/Diagnostic.h File Reference

Data Structures

- struct [Diagnostic](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Diagnostic](#) * [DIAGNOSTICHANDLE](#)

Functions

- [EXTERN_C CCAUXDLL_API](#)
[DIAGNOSTICHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetDiagnostic](#) (void)

7.12.1 Typedef Documentation

7.12.1.1 typedef [CrossControl::Diagnostic](#)* [DIAGNOSTICHANDLE](#)

7.12.2 Function Documentation

7.12.2.1 [EXTERN_C CCAUXDLL_API](#) [DIAGNOSTICHANDLE](#) [CCAUXDLL_CALLING_CONV](#)
[GetDiagnostic](#) (void)

Factory function that creates instances of the Diagnostic object.

Returns

[DIAGNOSTICHANDLE](#) to an allocated [Diagnostic](#) structure. The returned handle needs to be deallocated using the [Diagnostic::Release\(\)](#) method when it's no longer needed. Returns [NULL](#) if it fails to allocate memory.

Example Usage:

```

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

diagnostic_example(pDiagnostic);

pDiagnostic->Release();

```

7.13 IncludeFiles/DiagnosticCodes.h File Reference

Namespaces

- namespace [CrossControl](#)

Enumerations

- enum [startupReasonCodes](#) {
[startupReasonCodeUndefined](#) = 0x0000, [startupReasonCodeButtonPress](#) = 0x0055,
[startupReasonCodeExtCtrl](#) = 0x00AA, [startupReasonCodeMPRestart](#) = 0x00F0,
[startupReasonCodePowerOnStartup](#) = 0x000F }
- enum [shutDownReasonCodes](#) { [shutdownReasonCodeNoError](#) = 0x001F }
- enum [hwErrorStatusCodes](#) {
[errCodeNoErr](#) = 0, [errCodeFPGACONFReadErr](#) = 1, [errCodeFPGACONFUnexpVal](#) = 2, [errCodeCBRESETReadErr](#) = 3,
[errCodeSUS3ReadErr](#) = 4, [errCodeSUS4ReadErr](#) = 5, [errCodeSUS5ReadErr](#) = 6, [errCodePG5VSTBYReadErr](#) = 7,
[errCodePG5VSTBYUnexpVal](#) = 8, [errCodeCANPWROKReadErr](#) = 9, [errCodeVIDPWROKReadErr](#) = 10, [errCodeLVDSBLENReadErr](#) = 11,
[errCodeLVDSVDDENReadErr](#) = 12, [errCodeEXTCTRLONReadErr](#) = 13, [errCodeFPBTNONReadErr](#) = 14, [errCode24VReadErr](#) = 15,
[errCode24VOutOfLimits](#) = 16, [errCode24VINReadErr](#) = 17, [errCode24VINOutOfLimits](#) = 18, [errCode12VReadErr](#) = 19,
[errCode12VOutOfLimits](#) = 20, [errCode12VVIDEORReadErr](#) = 21, [errCode12VVIDEOOutOfLimits](#) = 22, [errCode5VSTBYReadErr](#) = 23,
[errCode5VSTBYOutOfLimits](#) = 24, [errCode5VReadErr](#) = 25, [errCode5VOutOfLimits](#) = 26, [errCode3V3ReadErr](#) = 27,
[errCode3V3OutOfLimits](#) = 28, [errCodeTFTVOLReadErr](#) = 29, [errCodeTFTVLOOutOfLimits](#) = 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, [errCodeSuspToRAMUnexpE-](#)

```

EPROMVal = 53, errCodeCANPWRUnexpEEPROMVal = 54, errCodeVID1P-
WRUnexpEEPROMVal = 55,
errCodeVID2PWRUnexpEEPROMVal = 56, errCodeVID3PWRUnexpEEPRO-
MVal = 57, errCodeVID4PWRUnexpEEPROMVal = 58, errCodeEXTFANUnexp-
EEPROMVal = 59,
errCodeLEDUnexpEEPROMVal = 60, errCodeUnitTypeUnexpEEPROMVal =
61, errCodeBLTYPEReadErrEEPROM = 62, errCodeFPBTNReadErrEEPRO-
M = 63,
errCodeEXTCTRLReadErrEEPROM = 64, errCodeMaxSuspTimeReadErrEE-
PROM = 65, errCodeLowRange24VReadErrEEPROM = 66, errCodeSuspToR-
AMReadErrEEPROM = 67,
errCodeCANPWRReadErrEEPROM = 68, errCodeVID1PWRReadErrEEPRO-
M = 69, errCodeVID2PWRReadErrEEPROM = 70, errCodeVID3PWRRead-
ErrEEPROM = 71,
errCodeVID4PWRReadErrEEPROM = 72, errCodeEXTFANReadErrEEPRO-
M = 73, errCodeLEDReadErrEEPROM = 74, errCodeUnitTypeReadErrEEPR-
OM = 75,
errCodeRCCInit = 76, errCodeDriverInit = 77, errCodeSetSUPPLYRESET =
78, errCodeRelSUPPLYRESET = 79,
errCodeSetSYSRESET = 80, errCodeRelSYSRESET = 81, errCodeSetPWRB-
TN = 82, errCodeRelPWRBTN = 83,
errCodeOnBL = 84, errCodeOffBL = 85, errCodeEXTFANOn = 86, errCodeE-
XTFANOff = 87,
errCodePWRENOn = 88, errCodePWRENOff = 89, errCodeMPPWRENOn =
90, errCodeMPPWRENOff = 91,
errCodeCANPWRENOn = 92, errCodeCANPWRENOff = 93, errCodeVID1P-
WRENOn = 94, errCodeVID1PWRENOff = 95,
errCodeVID2PWRENOn = 96, errCodeVID2PWRENOff = 97, errCodeVID3P-
WRENOn = 98, errCodeVID3PWRENOff = 99,
errCodeVID4PWRENOn = 100, errCodeVID4PWRENOff = 101, errCodeHE-
ATACTOn = 102, errCodeHEATACTOff = 103,
errCodeSetLEDCol = 104, errCodeSetLEDFreq = 105, errCodeManageLED =
106, errCodeManageCANPwr = 107,
errCodeManageMPPwr = 108, errCodeManageVidPwr = 109, errCodeManage-
PowSup = 110, errCodeManageReset = 111,
errCodeSSState = 112, errCodeVarWrapAround = 113, errCodeFPBTNUexp-
Val = 114, errCodeEXTCTRLUnexpVal = 115,
errCodeMAINPWROKReadErr = 116, errCodeFRONTSPAREReadErr = 117,
errCodeTIMERReadErr = 118, errCodeManageDiagnostics = 119,
errCodeFPBTNTimOutReadErrEEPROM = 120, errCodeEXTCTRLTimOutRead-
ErrEEPROM = 121, errCodeFPBTNAndExtCtrlDisabled = 122, errCodeSW-
VerReadErr = 123,
errCodeSWVerWriteErr = 124, errCodeManageActDeAct = 125, errCodeTick-
TimeOutTimer = 126, errCodeOperateModeStateError = 127,
errCodeHeatingTempReadErrEEPROM = 128, errCodeMPFailedStart = 129, err-
CodeReadErrEEPROM = 130, errCodeTimeOutWaitingForVoltages = 131,
errCodeMAX }

```

Functions

- 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)

7.14 IncludeFiles/DigIO.h File Reference

Data Structures

- struct [DigIO](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::DigIO](#) * [DIGIOHANDLE](#)

Functions

- EXTERN_C CCAUXDLL_API
[DIGIOHANDLE](#)
CCAUXDLL_CALLING_CONV [GetDigIO](#) (void)

Variables

- 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)

7.14.1 Typedef Documentation

7.14.1.1 typedef `CrossControl::DigIO*` DIGIOHANDLE

7.14.2 Function Documentation

7.14.2.1 EXTERN_C CCAUXDLL_API DIGIOHANDLE CCAUXDLL_CALLING_CONV GetDigIO (void)

Factory function that creates instances of the DigIO object.

Returns

DIGIOHANDLE to an allocated DigIO structure. The returned handle needs to be deallocated using the DigIO::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
DIGIOHANDLE pDigIO = ::GetDigIO();  
assert (pDigIO);  
  
list_digital_inputs (pDigIO);  
  
pDigIO->Release();
```

7.15 IncludeFiles/FirmwareUpgrade.h File Reference

Data Structures

- struct [FirmwareUpgrade](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef
[CrossControl::FirmwareUpgrade *](#) FIRMWAREUPGHANDLE

Functions

- EXTERN_C CCAUXDLL_API
[FIRMWAREUPGHANDLE](#)
CCAUXDLL_CALLING_CONV [GetFirmwareUpgrade](#) (void)

7.15.1 Typedef Documentation

7.15.1.1 `typedef CrossControl::FirmwareUpgrade* FIRMWAREUPGHANDLE`

7.15.2 Function Documentation

7.15.2.1 `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);
```

7.16 IncludeFiles/FrontLED.h File Reference

Data Structures

- struct [FrontLED](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::FrontLED](#) * [FRONTLEDHANDLE](#)

Functions

- `EXTERN_C CCAUXDLL_API
FRONTLEDHANDLE
CCAUXDLL_CALLING_CONV GetFrontLED (void)`

7.16.1 Typedef Documentation

7.16.1.1 typedef `CrossControl::FrontLED*` `FRONTLEDHANDLE`

7.16.2 Function Documentation

7.16.2.1 `EXTERN_C CCAUXDLL_API FRONTLEDHANDLE CCAUXDLL_CALLING_CONV GetFrontLED (void)`

Factory function that creates instances of the `FrontLED` object.

Returns

`FRONTLEDHANDLE` to an allocated `FrontLED` structure. The returned handle needs to be deallocated using the `FrontLED::Release()` method when it's no longer needed. Returns `NULL` if it fails to allocate memory.

Example Usage:

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

led_example(pFrontLED);

pFrontLED->Release();
```

7.17 IncludeFiles/Lightsensor.h File Reference

Data Structures

- struct [Lightsensor](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Lightsensor](#) * [LIGHTSENSORHANDLE](#)

Functions

- `EXTERN_C CCAUXDLL_API`
[LIGHTSENSORHANDLE](#)
`CCAUXDLL_CALLING_CONV` [GetLightsensor](#) (void)

7.17.1 Typedef Documentation

7.17.1.1 typedef `CrossControl::Lightsensor*` LIGHTSENSORHANDLE

7.17.2 Function Documentation

7.17.2.1 EXTERN_C CCAUXDLL_API LIGHTSENSORHANDLE
CCAUXDLL_CALLING_CONV GetLightsensor (void)

Factory function that creates instances of the Lightsensor object.

Returns

LIGHTSENSORHANDLE to an allocated Lightsensor structure. The returned handle needs to be deallocated using the Lightsensor::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
LIGHTSENSORHANDLE pLightSensor = ::GetLightsensor();  
assert (pLightSensor);  
  
ls_example (pLightSensor);  
  
pLightSensor->Release();
```

7.18 IncludeFiles/Power.h File Reference

Data Structures

- struct [Power](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Power](#) * POWERHANDLE

Functions

- EXTERN_C CCAUXDLL_API
[POWERHANDLE](#)
CCAUXDLL_CALLING_CONV [GetPower](#) (void)

7.18.1 Typedef Documentation

7.18.1.1 typedef CrossControl::Power* POWERHANDLE

7.18.2 Function Documentation

7.18.2.1 EXTERN_C CCAUXDLL_API POWERHANDLE CCAUXDLL_CALLING_CONV GetPower (void)

Factory function that creates instances of the Power object.

Returns

POWERHANDLE to an allocated Power structure. The returned handle needs to be deallocated using the Power::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

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

power_example(pPower);

pPower->Release();
```

7.19 IncludeFiles/PowerMgr.h File Reference

Data Structures

- struct [PowerMgr](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef enum [CrossControl::PowerMgrConf _PowerMgrConf](#)
- typedef enum [CrossControl::PowerMgrStatus _PowerMgrStatus](#)
- typedef [CrossControl::PowerMgr * POWERMGRHANDLE](#)

Enumerations

- enum [PowerMgrConf](#) { Normal = 0, ApplicationControlled = 1, BatterySuspend = 2 }

- enum `PowerMgrStatus` { `NoRequestsPending` = 0, `SuspendPending` = 1, `ShutdownPending` = 2 }

Functions

- EXTERN_C CCAUXDLL_API
POWERMGRHANDLE
CCAUXDLL_CALLING_CONV `GetPowerMgr` (void)

7.19.1 Typedef Documentation

- 7.19.1.1 typedef `CrossControl::PowerMgr* POWERMGRHANDLE`

7.19.2 Function Documentation

- 7.19.2.1 EXTERN_C CCAUXDLL_API POWERMGRHANDLE CCAUXDLL_CALLING_CONV
`GetPowerMgr` (void)

Factory function that creates instances of the `PowerMgr` object.

Returns

POWERMGRHANDLE to an allocated `PowerMgr` structure. The returned handle needs to be deallocated using the `PowerMgr::Release()` method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```

CrossControl::eErr err;
POWERMGRHANDLE pPowerMgr = ::GetPowerMgr();
BATTERYHANDLE pBattery = ::GetBattery();

assert(pPowerMgr);
assert(pBattery);

// Register a separate exit handler for the case where OS is initiating the shutdown. The Application
// must handle this case itself.
atexit(fnExit);

bool bBatt = false;
pBattery->isBatteryPresent(&bBatt);
if(bBatt) // Ask user wich configuration to use...
    cout << "Choose configuration to use, 0 - Normal, 1 - Application Controlled , 2 - Battery Suspend" <<
        endl;
else
    cout << "Choose configuration to use, 0 - Normal, 1 - Application Controlled" << endl;

cin >> suspendConfiguration;
pBattery->Release();

// Register that this application needs to delay suspend/shutdown
// This should be done as soon as possible.
// Then the app must poll getPowerMgrStatus() and allow the suspend/shutdown with
// setAppReadyForSuspendOrShutdown().
// Depending on application design, this might be best handled in a separate thread.
err = pPowerMgr->registerControlledSuspendOrShutDown((
    PowerMgrConf) suspendConfiguration);

```

```

if(err == ERR_SUCCESS)
    cout << "Registered to powerMgr." << endl;
else
    cout << "Error(" << err << ") in function registerControlledSuspendOrShutDown: " <<
        GetErrorStringA(err) << endl;

test_powermgr (pPowerMgr);

pPowerMgr->Release();

```

7.20 IncludeFiles/Releasenotes.dox File Reference

7.21 IncludeFiles/Smart.h File Reference

Data Structures

- struct [Smart](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Smart](#) * [SMARTHANDLE](#)

Functions

- [EXTERN_C](#) [CCAUXDLL_API](#)
[SMARTHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetSmart](#) (void)

7.21.1 Typedef Documentation

7.21.1.1 typedef [CrossControl::Smart](#)* [SMARTHANDLE](#)

7.21.2 Function Documentation

7.21.2.1 [EXTERN_C](#) [CCAUXDLL_API](#) [SMARTHANDLE](#) [CCAUXDLL_CALLING_CONV](#) [GetSmart](#) (void)

Factory function that creates instances of the Smart object.

Returns

SMARTHANDLE to an allocated AuxVersion structure. The returned handle needs to be deallocated using the Smart::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
SMARTHANDLE pSmart = ::GetSmart();
assert (pSmart);

show_card_data (pSmart);

pSmart->Release();
```

7.22 IncludeFiles/Telematics.h File Reference**Data Structures**

- struct [Telematics](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Telematics](#) * [TELEMATICSHANDLE](#)

Functions

- [EXTERN_C CCAUXDLL_API](#)
[TELEMATICSHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetTelematics](#) (void)

7.22.1 Typedef Documentation

7.22.1.1 typedef [CrossControl::Telematics](#)* [TELEMATICSHANDLE](#)

7.22.2 Function Documentation

7.22.2.1 [EXTERN_C CCAUXDLL_API](#) [TELEMATICSHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetTelematics](#) (void)

Factory function that creates instances of the Telematics object.

Returns

TELEMATICSHANDLE to an allocated Telematics structure. The returned handle needs to be deallocated using the Telematics::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

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

telematics_example(pTelematics);

pTelematics->Release();
```

7.23 IncludeFiles/TouchScreen.h File Reference**Data Structures**

- struct [TouchScreen](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::TouchScreen](#) * [TOUCHSCREENHANDLE](#)

Enumerations

- 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_CALIBRATION_WIDTH](#) = 9, [TS_CALIBRATION_MEASUREMENTS](#) = 10, [TS_RESTORE_DEFAULT_SETTINGS](#) = 11 }

Functions

- [EXTERN_C](#) [CCAUXDLL_API](#)
[TOUCHSCREENHANDLE](#)
[CCAUXDLL_CALLING_CONV](#) [GetTouchScreen](#) (void)

7.23.1 Typedef Documentation

7.23.1.1 typedef `CrossControl::TouchScreen*` TOUCHSCREENHANDLE

7.23.2 Function Documentation

7.23.2.1 EXTERN_C CCAUXDLL_API TOUCHSCREENHANDLE
CCAUXDLL_CALLING_CONV GetTouchScreen (void)

Factory function that creates instances of the TouchScreen object.

Returns

TOUCHSCREENHANDLE to an allocated TouchScreen structure. The returned handle needs to be deallocated using the TouchScreen::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

Example Usage:

```
TOUCHSCREENHANDLE pTouchScreen = ::GetTouchScreen();
assert (pTouchScreen);

touchscreen_example (pTouchScreen);

pTouchScreen->Release();
```

7.24 IncludeFiles/TouchScreenCalib.h File Reference

Data Structures

- struct [TouchScreenCalib](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef
[CrossControl::TouchScreenCalib](#) * TOUCHSCREENCALIBHANDLE

Enumerations

- enum [CalibrationModeSettings](#) {
 MODE_UNKNOWN = 0, MODE_NORMAL = 1, MODE_CALIBRATION_
 5P = 2, MODE_CALIBRATION_9P = 3,
 MODE_CALIBRATION_13P = 4 }

- enum [CalibrationConfigParam](#) {
[CONFIG_CALIBRATION_WITH](#) = 0, [CONFIG_CALIBRATION_MEASUREMENTS](#) = 1, [CONFIG_5P_CALIBRATION_POINT_BORDER](#) = 2, [CONFIG_13P_CALIBRATION_POINT_BORDER](#) = 3,
[CONFIG_13P_CALIBRATION_TRANSITION_MIN](#) = 4, [CONFIG_13P_CALIBRATION_TRANSITION_MAX](#) = 5 }

Functions

- [EXTERN_C CCAUXDLL_API TOUCHSCREENCALIBHANDLE CCAUXDLL_CALLING_CONV GetTouchScreenCalib](#) (void)

7.24.1 Typedef Documentation

- 7.24.1.1 [typedef CrossControl::TouchScreenCalib* TOUCHSCREENCALIBHANDLE](#)

7.24.2 Function Documentation

- 7.24.2.1 [EXTERN_C CCAUXDLL_API TOUCHSCREENCALIBHANDLE CCAUXDLL_CALLING_CONV GetTouchScreenCalib](#) (void)

Factory function that creates instances of the TouchScreenCalib object.

Returns

[TOUCHSCREENCALIBHANDLE](#) to an allocated TouchScreenCalib structure. The returned handle needs to be deallocated using the TouchScreenCalib::Release() method when it's no longer needed. Returns NULL if it fails to allocate memory.

7.25 IncludeFiles/Video.h File Reference

Data Structures

- struct [Video](#)

Namespaces

- namespace [CrossControl](#)

Typedefs

- typedef [CrossControl::Video](#) * [VIDEOHANDLE](#)

Functions

- EXTERN_C CCAUXDLL_API
VIDEOHANDLE
CCAUXDLL_CALLING_CONV [GetVideo](#) (void)

7.25.1 Typedef Documentation

7.25.1.1 typedef `CrossControl::Video*` VIDEOHANDLE

7.25.2 Function Documentation

7.25.2.1 EXTERN_C CCAUXDLL_API VIDEOHANDLE CCAUXDLL_CALLING_CONV
`GetVideo (void)`

Factory function that creates instances of the Video object.

Returns

VIDEOHANDLE to an allocated Video structure. The returned handle needs to be deallocated using the `Video::Release()` method when it's no longer needed. Returns NULL if it fails to allocate memory.

Index

`_PowerMgrConf`
 CrossControl, 10

`_PowerMgrStatus`
 CrossControl, 10

`ABOUTHANDLE`
 About.h, 176

`ADCHANDLE`
 Adc.h, 178

`AUXVERSIONHANDLE`
 AuxVersion.h, 179

About, 26

About.h
 ABOUTHANDLE, 176
 GetAbout, 176

Above80RunTime
 CrossControl::TimerType, 153

ackPowerRequest
 CrossControl::Power, 121

ActionShutDown
 CrossControl, 19

ActionSuspend
 CrossControl, 19

activateSnapshot
 CrossControl::Video, 164

Adc, 38

Adc.h
 ADCHANDLE, 178
 GetAdc, 178

Analog_Channel_1
 CrossControl, 22

Analog_Channel_2
 CrossControl, 22

Analog_Channel_3
 CrossControl, 22

Analog_Channel_4
 CrossControl, 22

ApplicationControlled
 CrossControl, 19

AuxVersion, 41

AuxVersion.h

AUXVERSIONHANDLE, 179
 GetAuxVersion, 179

BLUE
 CrossControl, 12

BPTS_BtnPressed
 CrossControl, 11

BPTS_BtnPressedLong
 CrossControl, 11

BPTS_No_Change
 CrossControl, 10

BPTS_Restart
 CrossControl, 11

BPTS_ShutDown
 CrossControl, 10

BPTS_SignalOff
 CrossControl, 11

BPTS_Suspend
 CrossControl, 10

BACKLIGHTHANDLE
 Backlight.h, 180

BATTERYHANDLE
 Battery.h, 181

BUZZERHANDLE
 Buzzer.h, 182

Backlight, 47

Backlight.h
 BACKLIGHTHANDLE, 180
 GetBacklight, 180

Battery, 53

Battery.h
 BATTERYHANDLE, 181
 GetBattery, 181

BatterySuspend
 CrossControl, 19

BatteryTimerType, 65

blue
 CrossControl::LedColorMixType, 114

Both_Button_And_Signal_Enabled
 CrossControl, 20

build

- CrossControl::version_info, 162
- ButtonPowerTransitionStatus
 - CrossControl, 10
- buzze
 - CrossControl::Buzzer, 68
- Buzzer, 67
- Buzzer.h
 - BUZZERHANDLE, 182
 - GetBuzzer, 182
- BuzzerSetup, 71
- CONFIG_13P_CALIBRATION_POINT_BORDER
 - CrossControl, 11
- CONFIG_13P_CALIBRATION_TRANSITION_MAX
 - CrossControl, 11
- CONFIG_13P_CALIBRATION_TRANSITION_MIN
 - CrossControl, 11
- CONFIG_5P_CALIBRATION_POINT_BORDER
 - CrossControl, 11
- CONFIG_CALIBRATION_MEASUREMENTS
 - CrossControl, 11
- CONFIG_CALIBRATION_WITH
 - CrossControl, 11
- CYAN
 - CrossControl, 12
- CANSETTINGHANDLE
 - CanSetting.h, 183
- CCAuxColor
 - CrossControl, 12
- CCStatus
 - CrossControl, 12
- CONFIGHANDLE
 - Config.h, 186
- CalibrationConfigParam
 - CrossControl, 11
- CalibrationModeSettings
 - CrossControl, 11
- CanFrameType
 - CrossControl, 11
- CanSetting, 72
- CanSetting.h
 - CANSETTINGHANDLE, 183
 - GetCanSetting, 183
- ChargingStatus_Charging
 - CrossControl, 12
- ChargingStatus_FullyCharged
 - CrossControl, 12
- ChargingStatus_NoCharge
 - CrossControl, 12
- ChargingStatus_TempHigh
 - CrossControl, 13
- ChargingStatus_TempLow
 - CrossControl, 12
- ChargingStatus_Unknown
 - CrossControl, 13
- ChargingStatus
 - CrossControl, 12
- CheckCalibrationPointFinished
 - CrossControl::TouchScreenCalib, 159
- clearHwErrorStatus
 - CrossControl::Diagnostic, 89
- Config, 75
- Config.h
 - CONFIGHANDLE, 186
 - GetConfig, 186
- createBitmap
 - CrossControl::Video, 164
- CrossControl
 - ActionShutDown, 19
 - ActionSuspend, 19
 - Analog_Channel_1, 22
 - Analog_Channel_2, 22
 - Analog_Channel_3, 22
 - Analog_Channel_4, 22
 - ApplicationControlled, 19
 - BLUE, 12
 - BPTS_BtnPressed, 11
 - BPTS_BtnPressedLong, 11
 - BPTS_No_Change, 10
 - BPTS_Restart, 11
 - BPTS_ShutDown, 10
 - BPTS_SignalOff, 11
 - BPTS_Suspend, 10
 - BatterySuspend, 19
 - Both_Button_And_Signal_Enabled, 20
 - CONFIG_13P_CALIBRATION_POINT_BORDER, 11
 - CONFIG_13P_CALIBRATION_TRANSITION_MAX, 11
 - CONFIG_13P_CALIBRATION_TRANSITION_MIN, 11
 - CONFIG_5P_CALIBRATION_POINT_BORDER, 11
 - CONFIG_CALIBRATION_MEASUREMENTS, 11

- CONFIG_CALIBRATION_WITH, 11
- CYAN, 12
- ChargingStatus_Charging, 12
- ChargingStatus_FullyCharged, 12
- ChargingStatus_NoCharge, 12
- ChargingStatus_TempHigh, 13
- ChargingStatus_TempLow, 12
- ChargingStatus_Unknown, 13
- DeInterlace_BOB, 13
- DeInterlace_Even, 13
- DeInterlace_Odd, 13
- Disabled, 12
- ERR_AVERAGE_CALC_STARTED, 13
- ERR_BUFFER_SIZE, 13
- ERR_CHECKSUM, 13
- ERR_CODE_NOT_EXIST, 13
- ERR_COMMAND_FAILED, 13
- ERR_CREATE_THREAD, 13
- ERR_DATATYPE_MISMATCH, 13
- ERR_DEVICE_READ_DATA_FAILED, 13
- ERR_DEVICE_WRITE_DATA_FAILED, 13
- ERR_EEPROM, 13
- ERR_I2C_EXPANDER_INIT_FAILED, 14
- ERR_I2C_EXPANDER_READ_FAILED, 14
- ERR_I2C_EXPANDER_WRITE_FAILED, 14
- ERR_IN_PROGRESS, 13
- ERR_INIT_FAILED, 13
- ERR_INVALID_DATA, 13
- ERR_INVALID_PARAMETER, 13
- ERR_IOCTL_FAILED, 13
- ERR_JIDA_TEMP, 13
- ERR_MEM_ALLOC_FAIL, 14
- ERR_NEWER_FPGA_VERSION_REQUIRED, 14
- ERR_NEWER_FRONT_VERSION_REQUIRED, 14
- ERR_NEWER_SS_VERSION_REQUIRED, 14
- ERR_NOT_RUNNING, 13
- ERR_NOT_SUPPORTED, 13
- ERR_OPEN_FAILED, 13
- ERR_SUCCESS, 13
- ERR_TELEMATICS_BT_NOT_AVAILABLE, 14
- ERR_TELEMATICS_GPRS_NOT_AVAILABLE, 14
- ERR_TELEMATICS_GPS_NOT_AVAILABLE, 14
- ERR_TELEMATICS_WLAN_NOT_AVAILABLE, 14
- ERR_UNKNOWN_FEATURE, 13
- ERR_VERIFY_FAILED, 13
- Enabled, 12
- errCode0V9OutOfLimits, 15
- errCode0V9ReadErr, 15
- errCode12VOutOfLimits, 15
- errCode12VReadErr, 15
- errCode12VVIDEOOutOfLimits, 15
- errCode12VVIDEOReadErr, 15
- errCode1V05OutOfLimits, 15
- errCode1V05ReadErr, 15
- errCode1V0OutOfLimits, 15
- errCode1V0ReadErr, 15
- errCode1V2OutOfLimits, 15
- errCode1V2ReadErr, 15
- errCode1V5OutOfLimits, 15
- errCode1V5ReadErr, 15
- errCode1V8OutOfLimits, 15
- errCode1V8ReadErr, 15
- errCode1V9OutOfLimits, 15
- errCode1V9ReadErr, 15
- errCode24VINOutOfLimits, 15
- errCode24VINReadErr, 15
- errCode24VOutOfLimits, 15
- errCode24VReadErr, 15
- errCode3V3OutOfLimits, 15
- errCode3V3ReadErr, 15
- errCode5VOutOfLimits, 15
- errCode5VReadErr, 15
- errCode5VSTBYOutOfLimits, 15
- errCode5VSTBYReadErr, 15
- errCodeBLTYPEReadErrEEPROM, 16
- errCodeBLTYPEUnexpEEPROMVal, 16
- errCodeCANPWRENOFF, 17
- errCodeCANPWRENOn, 17
- errCodeCANPWROKReadErr, 15
- errCodeCANPWRReadErrEEPROM, 16
- errCodeCANPWRUnexpEEPROMVal, 16
- errCodeCBRESETReadErr, 14
- errCodeDriverInit, 16
- errCodeEXTCTRLONReadErr, 15

- errCodeEXTCTRLReadErrEEPROM, 16
- errCodeEXTCTRLTimOutReadErrEEPROM, 17
- errCodeEXTCTRLUnexpEEPROMVal, 16
- errCodeEXTCTRLUnexpVal, 17
- errCodeEXTFANOff, 17
- errCodeEXTFANOn, 17
- errCodeEXTFANReadErrEEPROM, 16
- errCodeEXTFANUnexpEEPROMVal, 16
- errCodeFPBTNAndExtCtrlDisabled, 17
- errCodeFPBTNNonReadErr, 15
- errCodeFPBTNReadErrEEPROM, 16
- errCodeFPBNTimOutReadErrEEPROM, 17
- errCodeFPBTNUnexpEEPROMVal, 16
- errCodeFPBTNUnexpVal, 17
- errCodeFPGACONFReadErr, 14
- errCodeFPGACONFUnexpVal, 14
- errCodeFRONTSPAREReadErr, 17
- errCodeHEATACTOff, 17
- errCodeHEATACTOn, 17
- errCodeHeatingTempReadErrEEPROM, 18
- errCodeI2CTEMPOutOfLimits, 15
- errCodeI2CTEMPReadErr, 15
- errCodeLEDReadErrEEPROM, 16
- errCodeLEDUnexpEEPROMVal, 16
- errCodeLVDSBLENReadErr, 15
- errCodeLVDSVDDENReadErr, 15
- errCodeLowRange24VReadErrEEPROM, 16
- errCodeLowRange24VUnexpEEPROMVal, 16
- errCodeMAINPWROKReadErr, 17
- errCodeMAX, 18
- errCodeMPFailedStart, 18
- errCodeMPPWRENOFF, 17
- errCodeMPPWRENOn, 17
- errCodeManageActDeAct, 18
- errCodeManageCANPwr, 17
- errCodeManageDiagnostics, 17
- errCodeManageLED, 17
- errCodeManageMPPpwr, 17
- errCodeManagePowSup, 17
- errCodeManageReset, 17
- errCodeManageVidPwr, 17
- errCodeMaxSuspTimeReadErrEEPROM, 16
- errCodeNoErr, 14
- errCodeOffBL, 17
- errCodeOnBL, 16
- errCodeOperateModeStateError, 18
- errCodePG5VSTBYReadErr, 14
- errCodePG5VSTBYUnexpVal, 14
- errCodePWRENOFF, 17
- errCodePWRENOn, 17
- errCodeRCCInit, 16
- errCodeReadErrEEPROM, 18
- errCodeRelPWRBTN, 16
- errCodeRelSUPPLYRESET, 16
- errCodeRelSYSRESET, 16
- errCodeSSState, 17
- errCodeSTM32TEMPOutOfLimits, 16
- errCodeSTM32TEMPReadErr, 16
- errCodeSUS3ReadErr, 14
- errCodeSUS4ReadErr, 14
- errCodeSUS5ReadErr, 14
- errCodeSWVerReadErr, 18
- errCodeSWVerWriteErr, 18
- errCodeSetLEDCol, 17
- errCodeSetLEDFreq, 17
- errCodeSetPWRBTN, 16
- errCodeSetSUPPLYRESET, 16
- errCodeSetSYSRESET, 16
- errCodeSuspToRAMReadErrEEPROM, 16
- errCodeSuspToRAMUnexpEEPROMVal, 16
- errCodeTFTVLOOutOfLimits, 15
- errCodeTFTVOLReadErr, 15
- errCodeTIMERReadErr, 17
- errCodeTickTimeOutTimer, 18
- errCodeTimeOutWaitingForVoltages, 18
- errCodeUnitTypeReadErrEEPROM, 16
- errCodeUnitTypeUnexpEEPROMVal, 16
- errCodeVID1PWRENOFF, 17
- errCodeVID1PWRENOn, 17
- errCodeVID1PWRReadErrEEPROM, 16
- errCodeVID1PWRUnexpEEPROMVal, 16
- errCodeVID2PWRENOFF, 17
- errCodeVID2PWRENOn, 17

- errCodeVID2PWRReadErrEEPROM, 16
- errCodeVID2PWRUnexpEEPROMVal, 16
- errCodeVID3PWRENOff, 17
- errCodeVID3PWRENOOn, 17
- errCodeVID3PWRReadErrEEPROM, 16
- errCodeVID3PWRUnexpEEPROMVal, 16
- errCodeVID4PWRENOff, 17
- errCodeVID4PWRENOOn, 17
- errCodeVID4PWRReadErrEEPROM, 16
- errCodeVID4PWRUnexpEEPROMVal, 16
- errCodeVIDPWROKReadErr, 15
- errCodeVarWrapAround, 17
- ErrorStatus_ChargeFail, 14
- ErrorStatus_Init, 14
- ErrorStatus_NoError, 14
- ErrorStatus_Overcurrent, 14
- ErrorStatus_SecondaryTempSensor, 14
- ErrorStatus_ThermistorTempSensor, 14
- FrameExtended, 12
- FrameStandard, 12
- FrameStandardExtended, 12
- Front_Button_Enabled, 20
- GREEN, 12
- MAGENTA, 12
- MODE_CALIBRATION_13P, 11
- MODE_CALIBRATION_5P, 11
- MODE_CALIBRATION_9P, 11
- MODE_NORMAL, 11
- MODE_UNKNOWN, 11
- MOUSE_NEXT_BOOT, 20
- MOUSE_NOW, 20
- NoAction, 19
- NoRequestsPending, 19
- Normal, 19
- OnOff_Signal_Enabled, 20
- PowerSource_Battery, 19
- PowerSource_ExternalPower, 19
- RED, 12
- RangeExtended, 18
- RangeStandard, 18
- STD_B_D_G_H_I_N_PAL, 22
- STD_M_J_NTSC, 22
- STD_M_PAL, 22
- STD_NTSC, 22
- STD_PAL, 22
- STD_SECAM, 22
- SamplingModeAuto, 19
- SamplingModeExtended, 18
- SamplingModeStandard, 18
- ShutdownPending, 19
- shutdownReasonCodeNoError, 20
- startupReasonCodeButtonPress, 20
- startupReasonCodeExtCtrl, 20
- startupReasonCodeMPRestart, 20
- startupReasonCodePowerOnStartup, 20
- startupReasonCodeUndefined, 20
- SuspendPending, 19
- TEMP_BACKPLANE, 18
- TEMP_BOARD, 18
- TEMP_BOX, 18
- TEMP_CHIPSETS, 18
- TEMP_CPU, 18
- TEMP_ENV, 18
- TEMP_OTHER, 18
- TEMP_VIDEO, 18
- TOUCH_NEXT_BOOT, 20
- TOUCH_NOW, 20
- TS_CALIBRATION_MEASUREMENTS, 21
- TS_CALIBTATION_WIDTH, 21
- TS_DEBOUNCE_TIME, 21
- TS_DEBOUNCE_TIMEOUT_TIME, 21
- TS_DOUBLE_CLICK_TIME, 21
- TS_DOUBLECLICK_MAX_CLICK_TIME, 21
- TS_LOW_LEVEL, 21
- TS_MAX_RIGHTCLICK_DISTANCE, 21
- TS_RESTORE_DEFAULT_SETTINGS, 21
- TS_RIGHT_CLICK_TIME, 21
- TS_UNTOUCHLEVEL, 21
- TS_USE_DEJITTER, 21
- UNDEFINED_COLOR, 12
- UPGRADE_COMPLETE, 22
- UPGRADE_COMPLETE_WITH_ERRORS, 22
- UPGRADE_CONVERTING_FILE, 21
- UPGRADE_FLASHING, 22
- UPGRADE_INIT, 21
- UPGRADE_PREP_COM, 21
- UPGRADE_READING_FILE, 21
- UPGRADE_VERIFYING, 22

- VOLTAGE_0V9, 23
- VOLTAGE_12V, 22
- VOLTAGE_12VID, 22
- VOLTAGE_1V0, 23
- VOLTAGE_1V05, 23
- VOLTAGE_1V2, 23
- VOLTAGE_1V5, 23
- VOLTAGE_1V8, 23
- VOLTAGE_1V9, 23
- VOLTAGE_24V, 22
- VOLTAGE_24VIN, 22
- VOLTAGE_3V3, 22
- VOLTAGE_5V, 22
- VOLTAGE_5VSTB, 22
- VOLTAGE_VREF_INT, 23
- VOLTAGE_VTFT, 22
- YELLOW, 12
- CrossControl, 5
 - _PowerMgrConf, 10
 - _PowerMgrStatus, 10
 - ButtonPowerTransitionStatus, 10
 - CCAuxColor, 12
 - CCStatus, 12
 - CalibrationConfigParam, 11
 - CalibrationModeSettings, 11
 - CanFrameType, 11
 - ChargingStatus, 12
 - DeInterlaceMode, 13
 - DigitalIn_1, 25
 - DigitalIn_2, 25
 - DigitalIn_3, 25
 - DigitalIn_4, 25
 - eErr, 13
 - ErrorStatus, 14
 - GetErrorStringA, 23
 - GetErrorStringW, 23
 - GetHwErrorStatusStringA, 23
 - GetHwErrorStatusStringW, 24
 - GetStartupReasonStringA, 24
 - GetStartupReasonStringW, 24
 - hwErrorStatusCodes, 14
 - JidaSensorType, 18
 - LightSensorOperationRange, 18
 - LightSensorSamplingMode, 18
 - PowerAction, 19
 - PowerMgrConf, 19
 - PowerMgrStatus, 19
 - PowerSource, 19
 - shutDownReasonCodes, 19
 - startupReasonCodes, 20
 - TSAdvancedSettingsParameter, 20
 - TouchScreenModeSettings, 20
 - TriggerConf, 20
 - UpgradeAction, 21
 - VersionType, 10
 - Video1Conf, 25
 - Video2Conf, 25
 - Video3Conf, 25
 - Video4Conf, 25
 - VideoChannel, 22
 - videoStandard, 22
 - VoltageEnum, 22
- CrossControl::About
 - getAddOnHWversion, 29
 - getAddOnManufacturingDate, 29
 - getAddOnPCBArt, 29
 - getAddOnPCBSerial, 30
 - getIsBTMounted, 30
 - getIsDisplayAvailable, 31
 - getIsGPRSMounted, 31
 - getIsGPSPMounted, 31
 - getIsTouchScreenAvailable, 32
 - getIsWLANMounted, 32
 - getMainHWversion, 33
 - getMainManufacturingDate, 33
 - getMainPCBArt, 33
 - getMainPCBSerial, 34
 - getMainProdArtNr, 34
 - getMainProdRev, 35
 - getNrOfCANConnections, 35
 - getNrOfDigIOConnections, 35
 - getNrOfETHConnections, 36
 - getNrOfSerialConnections, 36
 - getNrOfUSBConnections, 37
 - getNrOfVideoConnections, 37
 - getUnitSerial, 38
 - Release, 38
- CrossControl::Adc
 - getVoltage, 40
 - Release, 40
- CrossControl::AuxVersion
 - getCCAuxDrvVersion, 43
 - getCCAuxVersion, 43
 - getFPGAVersion, 44
 - getFrontVersion, 45
 - getOSVersion, 45
 - getSSVersion, 46
 - Release, 46
- CrossControl::Backlight
 - getAutomaticBLFilter, 48

- getAutomaticBLParams, 49
- getAutomaticBLStatus, 49
- getIntensity, 49
- getLedDimming, 50
- getStatus, 50
- Release, 51
- setAutomaticBLFilter, 51
- setAutomaticBLParams, 52
- setIntensity, 52
- setLedDimming, 52
- startAutomaticBL, 53
- stopAutomaticBL, 53
- CrossControl::Battery
 - getBatteryChargingStatus, 58
 - getBatteryHWversion, 58
 - getBatterySerial, 59
 - getBatterySwVersion, 60
 - getBatteryTemp, 60
 - getBatteryVoltageStatus, 61
 - getHwErrorStatus, 61
 - getMinMaxTemp, 62
 - getPowerSource, 63
 - getTimer, 63
 - isBatteryPresent, 64
 - Release, 65
- CrossControl::BatteryTimerType
 - RunTime_0_40, 66
 - RunTime_40_60, 66
 - RunTime_60_70, 66
 - RunTime_70_80, 66
 - RunTime_Above80, 66
 - RunTime_m20, 66
 - RunTime_m20_0, 66
 - TotRunTimeBattery, 66
 - TotRunTimeMain, 66
- CrossControl::Buzzer
 - buzze, 68
 - getFrequency, 69
 - getTrigger, 69
 - getVolume, 69
 - Release, 70
 - setFrequency, 70
 - setTrigger, 70
 - setVolume, 71
- CrossControl::BuzzerSetup
 - frequency, 72
 - volume, 72
- CrossControl::CanSetting
 - getBaudrate, 73
 - getFrameType, 74
 - Release, 74
 - setBaudrate, 74
 - setFrameType, 75
- CrossControl::Config
 - getCanStartupPowerConfig, 77
 - getExtFanStartupPowerConfig, 77
 - getExtOnOffSigTrigTime, 78
 - getFrontBtnTrigTime, 78
 - getHeatingTempLimit, 78
 - getLongButtonPressAction, 79
 - getOnOffSigAction, 79
 - getPowerOnStartup, 80
 - getShortButtonPressAction, 80
 - getStartupTriggerConfig, 80
 - getStartupVoltageConfig, 81
 - getSuspendMaxTime, 81
 - getVideoStartupPowerConfig, 81
 - Release, 82
 - setCanStartupPowerConfig, 82
 - setExtFanStartupPowerConfig, 82
 - setExtOnOffSigTrigTime, 83
 - setFrontBtnTrigTime, 83
 - setHeatingTempLimit, 83
 - setLongButtonPressAction, 84
 - setOnOffSigAction, 84
 - setPowerOnStartup, 85
 - setShortButtonPressAction, 85
 - setStartupTriggerConfig, 85
 - setStartupVoltageConfig, 86
 - setSuspendMaxTime, 86
 - setVideoStartupPowerConfig, 87
- CrossControl::Diagnostic
 - clearHwErrorStatus, 89
 - getHwErrorStatus, 89
 - getMinMaxTemp, 89
 - getPCBTemp, 90
 - getPMTemp, 90
 - getPowerCycles, 90
 - getSSTemp, 91
 - getShutDownReason, 91
 - getStartupReason, 91
 - getTimer, 92
 - Release, 92
- CrossControl::DigIO
 - getDigIO, 93
 - getDigIODir, 94
 - Release, 94
 - setDigIO, 95
 - setDigIODir, 95
- CrossControl::FirmwareUpgrade

- getUpgradeStatus, 98
- Release, 99
- shutDown, 99
- startFpgaUpgrade, 99
- startFpgaVerification, 100
- startFrontUpgrade, 101
- startFrontVerification, 102
- startSSUpgrade, 103
- startSSVerification, 104
- CrossControl::FrontLED
 - getColor, 107, 108
 - getEnabledDuringStartup, 108
 - getIdleTime, 108
 - getNrOfPulses, 109
 - getOffTime, 109
 - getOnTime, 109
 - getSignal, 109
 - Release, 110
 - setColor, 110, 111
 - setEnabledDuringStartup, 111
 - setIdleTime, 111
 - setNrOfPulses, 112
 - setOff, 112
 - setOffTime, 112
 - setOnTime, 112
 - setSignal, 113
- CrossControl::LedColorMixType
 - blue, 114
 - green, 114
 - red, 114
- CrossControl::LedTimingType
 - idleTime, 114
 - nrOfPulses, 114
 - offTime, 114
 - onTime, 115
- CrossControl::Lightsensor
 - getAverageIlluminance, 116
 - getIlluminance, 117
 - getOperatingRange, 118
 - Release, 118
 - setOperatingRange, 118
 - startAverageCalc, 119
 - stopAverageCalc, 119
- CrossControl::Power
 - ackPowerRequest, 121
 - getBLPowerStatus, 122
 - getButtonPowerTransitionStatus, 122
 - getCanPowerStatus, 122
 - getExtFanPowerStatus, 123
 - getVideoPowerStatus, 123
 - Release, 123
 - setBLPowerStatus, 124
 - setCanPowerStatus, 124
 - setExtFanPowerStatus, 125
 - setVideoPowerStatus, 125
- CrossControl::PowerMgr
 - getConfiguration, 129
 - getPowerMgrStatus, 130
 - hasResumed, 132
 - registerControlledSuspendOrShutDown, 134
 - Release, 135
 - setAppReadyForSuspendOrShutdown, 135
- CrossControl::Smart
 - getDeviceSerial, 139
 - getInitialTime, 140
 - getRemainingLifeTime, 140
 - Release, 141
- CrossControl::Telematics
 - getBTPowerStatus, 145
 - getBTStartUpPowerStatus, 145
 - getGPRSPowerStatus, 146
 - getGPRSStartUpPowerStatus, 146
 - getGPSAntennaStatus, 147
 - getGPSPowerStatus, 147
 - getGPSStartUpPowerStatus, 148
 - getTelematicsAvailable, 148
 - getWLANPowerStatus, 149
 - getWLANStartUpPowerStatus, 149
 - Release, 150
 - setBTPowerStatus, 150
 - setBTStartUpPowerStatus, 151
 - setGPRSPowerStatus, 151
 - setGPRSStartUpPowerStatus, 151
 - setGPSPowerStatus, 151
 - setGPSStartUpPowerStatus, 152
 - setWLANPowerStatus, 152
 - setWLANStartUpPowerStatus, 152
- CrossControl::TimerType
 - Above80RunTime, 153
 - RunTime40_60, 153
 - RunTime60_70, 153
 - RunTime70_80, 153
 - TotHeatTime, 153
 - TotRunTime, 154
 - TotSuspTime, 154
- CrossControl::TouchScreen
 - getAdvancedSetting, 155
 - getMode, 156

- getMouseRightClickTime, 156
- Release, 157
- setAdvancedSetting, 157
- setMode, 158
- setMouseRightClickTime, 158
- CrossControl::TouchScreenCalib
 - CheckCalibrationPointFinished, 159
 - GetConfigParam, 159
 - GetMode, 160
 - Release, 160
 - SetCalibrationPoint, 160
 - SetConfigParam, 160
 - SetMode, 161
- CrossControl::UpgradeStatus
 - currentAction, 161
 - errorCode, 161
 - percent, 161
- CrossControl::Video
 - activateSnapshot, 164
 - createBitmap, 164
 - freeBmpBuffer, 164
 - getActiveChannel, 165
 - getColorKeys, 165
 - getCropping, 165
 - getDeInterlaceMode, 166
 - getDecoderReg, 166
 - getMirroring, 166
 - getRawImage, 167
 - getScaling, 167
 - getStatus, 167
 - getVideoArea, 168
 - getVideoStandard, 168
 - init, 169
 - minimize, 169
 - Release, 169
 - restore, 169
 - setActiveChannel, 169
 - setColorKeys, 170
 - setCropping, 170
 - setDeInterlaceMode, 171
 - setDecoderReg, 170
 - setMirroring, 171
 - setScaling, 172
 - setVideoArea, 172
 - showVideo, 172
 - takeSnapshot, 173
 - takeSnapshotBmp, 173
 - takeSnapshotRaw, 174
- CrossControl::received_video
 - received_framerate, 138
 - received_height, 138
 - received_width, 138
- CrossControl::version_info
 - build, 162
 - major, 162
 - minor, 162
 - release, 162
- CrossControl::video_dec_command
 - decoder_register, 174
 - register_value, 174
- currentAction
 - CrossControl::UpgradeStatus, 161
- DIAGNOSTICHANDLE
 - Diagnostic.h, 187
- DIGIOHANDLE
 - DigIO.h, 191
- DeInterlace_BOB
 - CrossControl, 13
- DeInterlace_Even
 - CrossControl, 13
- DeInterlace_Odd
 - CrossControl, 13
- DeInterlaceMode
 - CrossControl, 13
- decoder_register
 - CrossControl::video_dec_command, 174
- Diagnostic, 87
- Diagnostic.h
 - DIAGNOSTICHANDLE, 187
 - GetDiagnostic, 187
- DigIO, 92
- DigIO.h
 - DIGIOHANDLE, 191
 - GetDigIO, 191
- DigitalIn_1
 - CrossControl, 25
- DigitalIn_2
 - CrossControl, 25
- DigitalIn_3
 - CrossControl, 25
- DigitalIn_4
 - CrossControl, 25
- Disabled
 - CrossControl, 12
- ERR_AVERAGE_CALC_STARTED
 - CrossControl, 13
- ERR_BUFFER_SIZE
 - CrossControl, 13

- ERR_CHECKSUM
CrossControl, 13
- ERR_CODE_NOT_EXIST
CrossControl, 13
- ERR_COMMAND_FAILED
CrossControl, 13
- ERR_CREATE_THREAD
CrossControl, 13
- ERR_DATATYPE_MISMATCH
CrossControl, 13
- ERR_DEVICE_READ_DATA_FAILED
CrossControl, 13
- ERR_DEVICE_WRITE_DATA_FAILED
CrossControl, 13
- ERR_EEPROM
CrossControl, 13
- ERR_I2C_EXPANDER_INIT_FAILED
CrossControl, 14
- ERR_I2C_EXPANDER_READ_FAILED
CrossControl, 14
- ERR_I2C_EXPANDER_WRITE_FAILED
CrossControl, 14
- ERR_IN_PROGRESS
CrossControl, 13
- ERR_INIT_FAILED
CrossControl, 13
- ERR_INVALID_DATA
CrossControl, 13
- ERR_INVALID_PARAMETER
CrossControl, 13
- ERR_IOCTL_FAILED
CrossControl, 13
- ERR_JIDA_TEMP
CrossControl, 13
- ERR_MEM_ALLOC_FAIL
CrossControl, 14
- ERR_NEWER_FPGA_VERSION_REQUIRED
CrossControl, 14
- ERR_NEWER_FRONT_VERSION_REQUIRED
CrossControl, 14
- ERR_NEWER_SS_VERSION_REQUIRED
CrossControl, 14
- ERR_NOT_RUNNING
CrossControl, 13
- ERR_NOT_SUPPORTED
CrossControl, 13
- ERR_OPEN_FAILED
CrossControl, 13
- ERR_SUCCESS
CrossControl, 13
- ERR_TELEMATICS_BT_NOT_AVAILABLE
CrossControl, 14
- ERR_TELEMATICS_GPRS_NOT_AVAILABLE
CrossControl, 14
- ERR_TELEMATICS_GPS_NOT_AVAILABLE
CrossControl, 14
- ERR_TELEMATICS_WLAN_NOT_AVAILABLE
CrossControl, 14
- ERR_UNKNOWN_FEATURE
CrossControl, 13
- ERR_VERIFY_FAILED
CrossControl, 13
- eErr
CrossControl, 13
- Enabled
CrossControl, 12
- errCode0V9OutOfLimits
CrossControl, 15
- errCode0V9ReadErr
CrossControl, 15
- errCode12VOutOfLimits
CrossControl, 15
- errCode12VReadErr
CrossControl, 15
- errCode12VVIDEOOutOfLimits
CrossControl, 15
- errCode12VVIDEOReadErr
CrossControl, 15
- errCode1V05OutOfLimits
CrossControl, 15
- errCode1V05ReadErr
CrossControl, 15
- errCode1V0OutOfLimits
CrossControl, 15
- errCode1V0ReadErr
CrossControl, 15
- errCode1V2OutOfLimits
CrossControl, 15
- errCode1V2ReadErr
CrossControl, 15
- errCode1V5OutOfLimits
CrossControl, 15

- errCode1V5ReadErr
 - CrossControl, [15](#)
- errCode1V8OutOfLimits
 - CrossControl, [15](#)
- errCode1V8ReadErr
 - CrossControl, [15](#)
- errCode1V9OutOfLimits
 - CrossControl, [15](#)
- errCode1V9ReadErr
 - CrossControl, [15](#)
- errCode24VINOutOfLimits
 - CrossControl, [15](#)
- errCode24VINReadErr
 - CrossControl, [15](#)
- errCode24VOutOfLimits
 - CrossControl, [15](#)
- errCode24VReadErr
 - CrossControl, [15](#)
- errCode3V3OutOfLimits
 - CrossControl, [15](#)
- errCode3V3ReadErr
 - CrossControl, [15](#)
- errCode5VOutOfLimits
 - CrossControl, [15](#)
- errCode5VReadErr
 - CrossControl, [15](#)
- errCode5VSTBYOutOfLimits
 - CrossControl, [15](#)
- errCode5VSTBYReadErr
 - CrossControl, [15](#)
- errCodeBLTYPEReadErrEEPROM
 - CrossControl, [16](#)
- errCodeBLTYPEReadErrEEPROMVal
 - CrossControl, [16](#)
- errCodeCANPWRENOff
 - CrossControl, [17](#)
- errCodeCANPWRENOn
 - CrossControl, [17](#)
- errCodeCANPWROKReadErr
 - CrossControl, [15](#)
- errCodeCANPWRReadErrEEPROM
 - CrossControl, [16](#)
- errCodeCANPWRUnexpEEPROMVal
 - CrossControl, [16](#)
- errCodeCBRESETReadErr
 - CrossControl, [14](#)
- errCodeDriverInit
 - CrossControl, [16](#)
- errCodeEXTCTRLONReadErr
 - CrossControl, [15](#)
- errCodeEXTCTRLReadErrEEPROM
 - CrossControl, [16](#)
- errCodeEXTCTRLTimOutReadErrEEPROM
 - CrossControl, [17](#)
- errCodeEXTCTRLUnexpEEPROMVal
 - CrossControl, [16](#)
- errCodeEXTCTRLUnexpVal
 - CrossControl, [17](#)
- errCodeEXTFANOff
 - CrossControl, [17](#)
- errCodeEXTFANOn
 - CrossControl, [17](#)
- errCodeEXTFANReadErrEEPROM
 - CrossControl, [16](#)
- errCodeEXTFANUnexpEEPROMVal
 - CrossControl, [16](#)
- errCodeFPBTNAndExtCtrlDisabled
 - CrossControl, [17](#)
- errCodeFPBTNONReadErr
 - CrossControl, [15](#)
- errCodeFPBTNReadErrEEPROM
 - CrossControl, [16](#)
- errCodeFPBTNTimOutReadErrEEPROM
 - CrossControl, [17](#)
- errCodeFPBTNUnexpEEPROMVal
 - CrossControl, [16](#)
- errCodeFPBTNUnexpVal
 - CrossControl, [17](#)
- errCodeFPGACONFReadErr
 - CrossControl, [14](#)
- errCodeFPGACONFUnexpVal
 - CrossControl, [14](#)
- errCodeFRONTSPAREReadErr
 - CrossControl, [17](#)
- errCodeHEATACTOff
 - CrossControl, [17](#)
- errCodeHEATACTOn
 - CrossControl, [17](#)
- errCodeHeatingTempReadErrEEPROM
 - CrossControl, [18](#)
- errCodeI2CTEMPOutOfLimits
 - CrossControl, [15](#)
- errCodeI2CTEMPReadErr
 - CrossControl, [15](#)
- errCodeLEDReadErrEEPROM
 - CrossControl, [16](#)
- errCodeLEDUnexpEEPROMVal
 - CrossControl, [16](#)
- errCodeLVDSBLENReadErr

- CrossControl, 15
- errCodeLVDSVDDENReadErr
 - CrossControl, 15
- errCodeLowRange24VReadErrEEPROM
 - CrossControl, 16
- errCodeLowRange24VUnexpEEPROMVal
 - CrossControl, 16
- errCodeMAINPWOKReadErr
 - CrossControl, 17
- errCodeMAX
 - CrossControl, 18
- errCodeMPFailedStart
 - CrossControl, 18
- errCodeMPPWRENOff
 - CrossControl, 17
- errCodeMPPWRENOn
 - CrossControl, 17
- errCodeManageActDeAct
 - CrossControl, 18
- errCodeManageCANPwr
 - CrossControl, 17
- errCodeManageDiagnostics
 - CrossControl, 17
- errCodeManageLED
 - CrossControl, 17
- errCodeManageMPPPwr
 - CrossControl, 17
- errCodeManagePowSup
 - CrossControl, 17
- errCodeManageReset
 - CrossControl, 17
- errCodeManageVidPwr
 - CrossControl, 17
- errCodeMaxSuspTimeReadErrEEPROM
 - CrossControl, 16
- errCodeNoErr
 - CrossControl, 14
- errCodeOffBL
 - CrossControl, 17
- errCodeOnBL
 - CrossControl, 16
- errCodeOperateModeStateError
 - CrossControl, 18
- errCodePG5VSTBYReadErr
 - CrossControl, 14
- errCodePG5VSTBYUnexpVal
 - CrossControl, 14
- errCodePWRENOff
 - CrossControl, 17
- errCodePWRENOn
 - CrossControl, 17
- errCodeRCCInit
 - CrossControl, 16
- errCodeReadErrEEPROM
 - CrossControl, 18
- errCodeRelPWRBTN
 - CrossControl, 16
- errCodeRelSUPPLYRESET
 - CrossControl, 16
- errCodeRelSYSRESET
 - CrossControl, 16
- errCodeSSState
 - CrossControl, 17
- errCodeSTM32TEMPOutOfLimits
 - CrossControl, 16
- errCodeSTM32TEMPReadErr
 - CrossControl, 16
- errCodeSUS3ReadErr
 - CrossControl, 14
- errCodeSUS4ReadErr
 - CrossControl, 14
- errCodeSUS5ReadErr
 - CrossControl, 14
- errCodeSWVerReadErr
 - CrossControl, 18
- errCodeSWVerWriteErr
 - CrossControl, 18
- errCodeSetLEDCol
 - CrossControl, 17
- errCodeSetLEDFreq
 - CrossControl, 17
- errCodeSetPWRBTN
 - CrossControl, 16
- errCodeSetSUPPLYRESET
 - CrossControl, 16
- errCodeSetSYSRESET
 - CrossControl, 16
- errCodeSuspToRAMReadErrEEPROM
 - CrossControl, 16
- errCodeSuspToRAMUnexpEEPROMVal
 - CrossControl, 16
- errCodeTFTVOLOutOfLimits
 - CrossControl, 15
- errCodeTFTVOLReadErr
 - CrossControl, 15
- errCodeTIMERReadErr
 - CrossControl, 17
- errCodeTickTimeOutTimer
 - CrossControl, 18
- errCodeTimeOutWaitingForVoltages

- CrossControl, 18
- errCodeUnitTypeReadErrEEPROM
 - CrossControl, 16
- errCodeUnitTypeUnexpEEPROMVal
 - CrossControl, 16
- errCodeVID1PWRENOff
 - CrossControl, 17
- errCodeVID1PWRENOOn
 - CrossControl, 17
- errCodeVID1PWRReadErrEEPROM
 - CrossControl, 16
- errCodeVID1PWRUnexpEEPROMVal
 - CrossControl, 16
- errCodeVID2PWRENOff
 - CrossControl, 17
- errCodeVID2PWRENOOn
 - CrossControl, 17
- errCodeVID2PWRReadErrEEPROM
 - CrossControl, 16
- errCodeVID2PWRUnexpEEPROMVal
 - CrossControl, 16
- errCodeVID3PWRENOff
 - CrossControl, 17
- errCodeVID3PWRENOOn
 - CrossControl, 17
- errCodeVID3PWRReadErrEEPROM
 - CrossControl, 16
- errCodeVID3PWRUnexpEEPROMVal
 - CrossControl, 16
- errCodeVID4PWRENOff
 - CrossControl, 17
- errCodeVID4PWRENOOn
 - CrossControl, 17
- errCodeVID4PWRReadErrEEPROM
 - CrossControl, 16
- errCodeVID4PWRUnexpEEPROMVal
 - CrossControl, 16
- errCodeVIDPWROKReadErr
 - CrossControl, 15
- errCodeVarWrapAround
 - CrossControl, 17
- ErrorStatus_ChargeFail
 - CrossControl, 14
- ErrorStatus_Init
 - CrossControl, 14
- ErrorStatus_NoError
 - CrossControl, 14
- ErrorStatus_Overcurrent
 - CrossControl, 14
- ErrorStatus_SecondaryTempSensor
 - CrossControl, 14
- ErrorStatus_ThermistorTempSensor
 - CrossControl, 14
- errorCode
 - CrossControl::UpgradeStatus, 161
- ErrorStatus
 - CrossControl, 14
- FIRMWAREUPGHANDLE
 - FirmwareUpgrade.h, 192
- FRONTLEDHANDLE
 - FrontLED.h, 193
- FirmwareUpgrade, 96
- FirmwareUpgrade.h
 - FIRMWAREUPGHANDLE, 192
 - GetFirmwareUpgrade, 192
- FrameExtended
 - CrossControl, 12
- FrameStandard
 - CrossControl, 12
- FrameStandardExtended
 - CrossControl, 12
- freeBmpBuffer
 - CrossControl::Video, 164
- frequency
 - CrossControl::BuzzerSetup, 72
- Front_Button_Enabled
 - CrossControl, 20
- FrontLED, 105
- FrontLED.h
 - FRONTLEDHANDLE, 193
 - GetFrontLED, 193
- GREEN
 - CrossControl, 12
- GetAbout
 - About.h, 176
- getActiveChannel
 - CrossControl::Video, 165
- GetAdc
 - Adc.h, 178
- getAddOnHWversion
 - CrossControl::About, 29
- getAddOnManufacturingDate
 - CrossControl::About, 29
- getAddOnPCBART
 - CrossControl::About, 29
- getAddOnPCBSerial
 - CrossControl::About, 30
- getAdvancedSetting

- CrossControl::TouchScreen, 155
- getAutomaticBLFilter
 - CrossControl::Backlight, 48
- getAutomaticBLParams
 - CrossControl::Backlight, 49
- getAutomaticBLStatus
 - CrossControl::Backlight, 49
- GetAuxVersion
 - AuxVersion.h, 179
- getAverageIlluminance
 - CrossControl::Lightsensor, 116
- getBLPowerStatus
 - CrossControl::Power, 122
- getBTPowerStatus
 - CrossControl::Telematics, 145
- getBTStartUpPowerStatus
 - CrossControl::Telematics, 145
- GetBacklight
 - Backlight.h, 180
- GetBattery
 - Battery.h, 181
- getBatteryChargingStatus
 - CrossControl::Battery, 58
- getBatteryHWversion
 - CrossControl::Battery, 58
- getBatterySerial
 - CrossControl::Battery, 59
- getBatterySwVersion
 - CrossControl::Battery, 60
- getBatteryTemp
 - CrossControl::Battery, 60
- getBatteryVoltageStatus
 - CrossControl::Battery, 61
- getBaudrate
 - CrossControl::CanSetting, 73
- getButtonPowerTransitionStatus
 - CrossControl::Power, 122
- GetBuzzer
 - Buzzer.h, 182
- getCCAuxDrvVersion
 - CrossControl::AuxVersion, 43
- getCCAuxVersion
 - CrossControl::AuxVersion, 43
- getCanPowerStatus
 - CrossControl::Power, 122
- GetCanSetting
 - CanSetting.h, 183
- getCanStartupPowerConfig
 - CrossControl::Config, 77
- getColor
 - CrossControl::FrontLED, 107, 108
- getColorKeys
 - CrossControl::Video, 165
- GetConfig
 - Config.h, 186
- GetConfigParam
 - CrossControl::TouchScreenCalib, 159
- getConfiguration
 - CrossControl::PowerMgr, 129
- getCropping
 - CrossControl::Video, 165
- getDeInterlaceMode
 - CrossControl::Video, 166
- getDecoderReg
 - CrossControl::Video, 166
- getDeviceSerial
 - CrossControl::Smart, 139
- GetDiagnostic
 - Diagnostic.h, 187
- GetDigIO
 - DigIO.h, 191
- getDigIO
 - CrossControl::DigIO, 93
- getDigIODir
 - CrossControl::DigIO, 94
- getEnabledDuringStartup
 - CrossControl::FrontLED, 108
- GetErrorStringA
 - CrossControl, 23
- GetErrorStringW
 - CrossControl, 23
- getExtFanPowerStatus
 - CrossControl::Power, 123
- getExtFanStartupPowerConfig
 - CrossControl::Config, 77
- getExtOnOffSigTrigTime
 - CrossControl::Config, 78
- getFPGAVersion
 - CrossControl::AuxVersion, 44
- GetFirmwareUpgrade
 - FirmwareUpgrade.h, 192
- getFrameType
 - CrossControl::CanSetting, 74
- getFrequency
 - CrossControl::Buzzer, 69
- getFrontBtnTrigTime
 - CrossControl::Config, 78
- GetFrontLED
 - FrontLED.h, 193
- getFrontVersion

- CrossControl::AuxVersion, 45
- getGPRSPowerStatus
 - CrossControl::Telematics, 146
- getGPRSStartupPowerStatus
 - CrossControl::Telematics, 146
- getGPSAntennaStatus
 - CrossControl::Telematics, 147
- getGPSPowerStatus
 - CrossControl::Telematics, 147
- getGPSStartupPowerStatus
 - CrossControl::Telematics, 148
- getHeatingTempLimit
 - CrossControl::Config, 78
- getHwErrorStatus
 - CrossControl::Battery, 61
 - CrossControl::Diagnostic, 89
- GetHwErrorStatusStringA
 - CrossControl, 23
- GetHwErrorStatusStringW
 - CrossControl, 24
- getIdleTime
 - CrossControl::FrontLED, 108
- getIlluminance
 - CrossControl::Lightsensor, 117
- getInitialTime
 - CrossControl::Smart, 140
- getIntensity
 - CrossControl::Backlight, 49
- getIsBTMounted
 - CrossControl::About, 30
- getIsDisplayAvailable
 - CrossControl::About, 31
- getIsGPRSMounted
 - CrossControl::About, 31
- getIsGPSMounted
 - CrossControl::About, 31
- getIsTouchScreenAvailable
 - CrossControl::About, 32
- getIsWLANMounted
 - CrossControl::About, 32
- getLedDimming
 - CrossControl::Backlight, 50
- GetLightsensor
 - Lightsensor.h, 194
- getLongButtonPressAction
 - CrossControl::Config, 79
- getMainHWversion
 - CrossControl::About, 33
- getMainManufacturingDate
 - CrossControl::About, 33
- getMainPCBArt
 - CrossControl::About, 33
- getMainPCBSerial
 - CrossControl::About, 34
- getMainProdArtNr
 - CrossControl::About, 34
- getMainProdRev
 - CrossControl::About, 35
- getMinMaxTemp
 - CrossControl::Battery, 62
 - CrossControl::Diagnostic, 89
- getMirroring
 - CrossControl::Video, 166
- GetMode
 - CrossControl::TouchScreenCalib, 160
- getMode
 - CrossControl::TouchScreen, 156
- getMouseRightClickTime
 - CrossControl::TouchScreen, 156
- getNrOfCANConnections
 - CrossControl::About, 35
- getNrOfDigIOConnections
 - CrossControl::About, 35
- getNrOfETHConnections
 - CrossControl::About, 36
- getNrOfPulses
 - CrossControl::FrontLED, 109
- getNrOfSerialConnections
 - CrossControl::About, 36
- getNrOfUSBConnections
 - CrossControl::About, 37
- getNrOfVideoConnections
 - CrossControl::About, 37
- getOSVersion
 - CrossControl::AuxVersion, 45
- getOffTime
 - CrossControl::FrontLED, 109
- getOnOffSigAction
 - CrossControl::Config, 79
- getOnTime
 - CrossControl::FrontLED, 109
- getOperatingRange
 - CrossControl::Lightsensor, 118
- getPCBTemp
 - CrossControl::Diagnostic, 90
- getPMTemp
 - CrossControl::Diagnostic, 90
- GetPower
 - Power.h, 195
- getPowerCycles

- CrossControl::Diagnostic, 90
- GetPowerMgr
 - PowerMgr.h, 196
- getPowerMgrStatus
 - CrossControl::PowerMgr, 130
- getPowerOnStartup
 - CrossControl::Config, 80
- getPowerSource
 - CrossControl::Battery, 63
- getRawImage
 - CrossControl::Video, 167
- getRemainingLifeTime
 - CrossControl::Smart, 140
- getSSTemp
 - CrossControl::Diagnostic, 91
- getSSVersion
 - CrossControl::AuxVersion, 46
- getScaling
 - CrossControl::Video, 167
- getShortButtonPressAction
 - CrossControl::Config, 80
- getShutDownReason
 - CrossControl::Diagnostic, 91
- getSignal
 - CrossControl::FrontLED, 109
- GetSmart
 - Smart.h, 197
- getStartupReason
 - CrossControl::Diagnostic, 91
- GetStartupReasonStringA
 - CrossControl, 24
- GetStartupReasonStringW
 - CrossControl, 24
- getStartupTriggerConfig
 - CrossControl::Config, 80
- getStartupVoltageConfig
 - CrossControl::Config, 81
- getStatus
 - CrossControl::Backlight, 50
 - CrossControl::Video, 167
- getSuspendMaxTime
 - CrossControl::Config, 81
- GetTelematics
 - Telematics.h, 198
- getTelematicsAvailable
 - CrossControl::Telematics, 148
- getTimer
 - CrossControl::Battery, 63
 - CrossControl::Diagnostic, 92
- GetTouchScreen
 - TouchScreen.h, 200
- GetTouchScreenCalib
 - TouchScreenCalib.h, 201
- getTrigger
 - CrossControl::Buzzer, 69
- getUnitSerial
 - CrossControl::About, 38
- getUpgradeStatus
 - CrossControl::FirmwareUpgrade, 98
- GetVideo
 - Video.h, 202
- getVideoArea
 - CrossControl::Video, 168
- getVideoPowerStatus
 - CrossControl::Power, 123
- getVideoStandard
 - CrossControl::Video, 168
- getVideoStartupPowerConfig
 - CrossControl::Config, 81
- getVoltage
 - CrossControl::Adc, 40
- getVolume
 - CrossControl::Buzzer, 69
- getWLANPowerStatus
 - CrossControl::Telematics, 149
- getWLANStartUpPowerStatus
 - CrossControl::Telematics, 149
- green
 - CrossControl::LedColorMixType, 114
- hasResumed
 - CrossControl::PowerMgr, 132
- hwErrorStatusCodes
 - CrossControl, 14
- idleTime
 - CrossControl::LedTimingType, 114
- IncludeFiles/About.h, 176
- IncludeFiles/Adc.h, 177
- IncludeFiles/AuxVersion.h, 178
- IncludeFiles/Backlight.h, 179
- IncludeFiles/Battery.h, 180
- IncludeFiles/Buzzer.h, 181
- IncludeFiles/CCAuxErrors.h, 183
- IncludeFiles/CCAuxTypes.h, 184
- IncludeFiles/CCPlatform.h, 185
- IncludeFiles/CanSetting.h, 182
- IncludeFiles/Config.h, 185
- IncludeFiles/Diagnostic.h, 187
- IncludeFiles/DiagnosticCodes.h, 188

- IncludeFiles/DigIO.h, 190
- IncludeFiles/FirmwareUpgrade.h, 191
- IncludeFiles/FrontLED.h, 192
- IncludeFiles/Lightsensor.h, 193
- IncludeFiles/Power.h, 194
- IncludeFiles/PowerMgr.h, 195
- IncludeFiles/Releasenotes.dox, 197
- IncludeFiles/Smart.h, 197
- IncludeFiles/Telematics.h, 198
- IncludeFiles/TouchScreen.h, 199
- IncludeFiles/TouchScreenCalib.h, 200
- IncludeFiles/Video.h, 201
- init
 - CrossControl::Video, 169
- isBatteryPresent
 - CrossControl::Battery, 64
- JidaSensorType
 - CrossControl, 18
- LIGHTSENSORHANDLE
 - Lightsensor.h, 194
 - LedColorMixType, 113
 - LedTimingType, 114
 - LightSensorOperationRange
 - CrossControl, 18
 - LightSensorSamplingMode
 - CrossControl, 18
 - Lightsensor, 115
 - Lightsensor.h
 - GetLightsensor, 194
 - LIGHTSENSORHANDLE, 194
- MAGENTA
 - CrossControl, 12
- MODE_CALIBRATION_13P
 - CrossControl, 11
- MODE_CALIBRATION_5P
 - CrossControl, 11
- MODE_CALIBRATION_9P
 - CrossControl, 11
- MODE_NORMAL
 - CrossControl, 11
- MODE_UNKNOWN
 - CrossControl, 11
- MOUSE_NEXT_BOOT
 - CrossControl, 20
- MOUSE_NOW
 - CrossControl, 20
- major
 - CrossControl::version_info, 162
- minimize
 - CrossControl::Video, 169
- minor
 - CrossControl::version_info, 162
- NoAction
 - CrossControl, 19
- NoRequestsPending
 - CrossControl, 19
- Normal
 - CrossControl, 19
- nrOfPulses
 - CrossControl::LedTimingType, 114
- offTime
 - CrossControl::LedTimingType, 114
- OnOff_Signal_Enabled
 - CrossControl, 20
- onTime
 - CrossControl::LedTimingType, 115
- POWERHANDLE
 - Power.h, 195
- POWERMGRHANDLE
 - PowerMgr.h, 196
- percent
 - CrossControl::UpgradeStatus, 161
- Power, 120
- Power.h
 - GetPower, 195
 - POWERHANDLE, 195
- PowerSource_Battery
 - CrossControl, 19
- PowerSource_ExternalPower
 - CrossControl, 19
- PowerAction
 - CrossControl, 19
- PowerMgr, 125
- PowerMgr.h
 - GetPowerMgr, 196
 - POWERMGRHANDLE, 196
- PowerMgrConf
 - CrossControl, 19
- PowerMgrStatus
 - CrossControl, 19
- PowerSource
 - CrossControl, 19
- RED

- CrossControl, 12
- RangeExtended
 - CrossControl, 18
- RangeStandard
 - CrossControl, 18
- received_framerate
 - CrossControl::received_video, 138
- received_height
 - CrossControl::received_video, 138
- received_video, 137
- received_width
 - CrossControl::received_video, 138
- red
 - CrossControl::LedColorMixType, 114
- register_value
 - CrossControl::video_dec_command, 174
- registerControlledSuspendOrShutDown
 - CrossControl::PowerMgr, 134
- Release
 - CrossControl::About, 38
 - CrossControl::Adc, 40
 - CrossControl::AuxVersion, 46
 - CrossControl::Backlight, 51
 - CrossControl::Battery, 65
 - CrossControl::Buzzer, 70
 - CrossControl::CanSetting, 74
 - CrossControl::Config, 82
 - CrossControl::Diagnostic, 92
 - CrossControl::DigIO, 94
 - CrossControl::FirmwareUpgrade, 99
 - CrossControl::FrontLED, 110
 - CrossControl::Lightsensor, 118
 - CrossControl::Power, 123
 - CrossControl::PowerMgr, 135
 - CrossControl::Smart, 141
 - CrossControl::Telematics, 150
 - CrossControl::TouchScreen, 157
 - CrossControl::TouchScreenCalib, 160
 - CrossControl::Video, 169
- release
 - CrossControl::version_info, 162
- restore
 - CrossControl::Video, 169
- RunTime40_60
 - CrossControl::TimerType, 153
- RunTime60_70
 - CrossControl::TimerType, 153
- RunTime70_80
 - CrossControl::TimerType, 153
- RunTime_0_40
 - CrossControl::BatteryTimerType, 66
- RunTime_40_60
 - CrossControl::BatteryTimerType, 66
- RunTime_60_70
 - CrossControl::BatteryTimerType, 66
- RunTime_70_80
 - CrossControl::BatteryTimerType, 66
- RunTime_Above80
 - CrossControl::BatteryTimerType, 66
- RunTime_m20
 - CrossControl::BatteryTimerType, 66
- RunTime_m20_0
 - CrossControl::BatteryTimerType, 66
- STD_B_D_G_H_I_N_PAL
 - CrossControl, 22
- STD_M_J_NTSC
 - CrossControl, 22
- STD_M_PAL
 - CrossControl, 22
- STD_NTSC
 - CrossControl, 22
- STD_PAL
 - CrossControl, 22
- STD_SECAM
 - CrossControl, 22
- SMARTHANDLE
 - Smart.h, 197
- SamplingModeAuto
 - CrossControl, 19
- SamplingModeExtended
 - CrossControl, 18
- SamplingModeStandard
 - CrossControl, 18
- setActiveChannel
 - CrossControl::Video, 169
- setAdvancedSetting
 - CrossControl::TouchScreen, 157
- setAppReadyForSuspendOrShutdown
 - CrossControl::PowerMgr, 135
- setAutomaticBLFilter
 - CrossControl::Backlight, 51
- setAutomaticBLParams
 - CrossControl::Backlight, 52
- setBLPowerStatus
 - CrossControl::Power, 124
- setBTPowerStatus
 - CrossControl::Telematics, 150
- setBTStartUpPowerStatus
 - CrossControl::Telematics, 151

- setBaudrate
 - CrossControl::CanSetting, 74
- SetCalibrationPoint
 - CrossControl::TouchScreenCalib, 160
- setCanPowerStatus
 - CrossControl::Power, 124
- setCanStartupPowerConfig
 - CrossControl::Config, 82
- setColor
 - CrossControl::FrontLED, 110, 111
- setColorKeys
 - CrossControl::Video, 170
- SetConfigParam
 - CrossControl::TouchScreenCalib, 160
- setCropping
 - CrossControl::Video, 170
- setDeInterlaceMode
 - CrossControl::Video, 171
- setDecoderReg
 - CrossControl::Video, 170
- setDigIO
 - CrossControl::DigIO, 95
- setDigIODir
 - CrossControl::DigIO, 95
- setEnabledDuringStartup
 - CrossControl::FrontLED, 111
- setExtFanPowerStatus
 - CrossControl::Power, 125
- setExtFanStartupPowerConfig
 - CrossControl::Config, 82
- setExtOnOffSigTrigTime
 - CrossControl::Config, 83
- setFrameType
 - CrossControl::CanSetting, 75
- setFrequency
 - CrossControl::Buzzer, 70
- setFrontBtnTrigTime
 - CrossControl::Config, 83
- setGPRSPowerStatus
 - CrossControl::Telematics, 151
- setGPRSStartUpPowerStatus
 - CrossControl::Telematics, 151
- setGPSPowerStatus
 - CrossControl::Telematics, 151
- setGPSStartUpPowerStatus
 - CrossControl::Telematics, 152
- setHeatingTempLimit
 - CrossControl::Config, 83
- setIdleTime
 - CrossControl::FrontLED, 111
- setIntensity
 - CrossControl::Backlight, 52
- setLedDimming
 - CrossControl::Backlight, 52
- setLongButtonPressAction
 - CrossControl::Config, 84
- setMirroring
 - CrossControl::Video, 171
- SetMode
 - CrossControl::TouchScreenCalib, 161
- setMode
 - CrossControl::TouchScreen, 158
- setMouseRightClickTime
 - CrossControl::TouchScreen, 158
- setNrOfPulses
 - CrossControl::FrontLED, 112
- setOff
 - CrossControl::FrontLED, 112
- setOffTime
 - CrossControl::FrontLED, 112
- setOnOffSigAction
 - CrossControl::Config, 84
- setOnTime
 - CrossControl::FrontLED, 112
- setOperatingRange
 - CrossControl::Lightsensor, 118
- setPowerOnStartup
 - CrossControl::Config, 85
- setScaling
 - CrossControl::Video, 172
- setShortButtonPressAction
 - CrossControl::Config, 85
- setSignal
 - CrossControl::FrontLED, 113
- setStartupTriggerConfig
 - CrossControl::Config, 85
- setStartupVoltageConfig
 - CrossControl::Config, 86
- setSuspendMaxTime
 - CrossControl::Config, 86
- setTrigger
 - CrossControl::Buzzer, 70
- setVideoArea
 - CrossControl::Video, 172
- setVideoPowerStatus
 - CrossControl::Power, 125
- setVideoStartupPowerConfig
 - CrossControl::Config, 87
- setVolume
 - CrossControl::Buzzer, 71

- setWLANPowerStatus
 - CrossControl::Telematics, 152
- setWLANStartUpPowerStatus
 - CrossControl::Telematics, 152
- showVideo
 - CrossControl::Video, 172
- shutDown
 - CrossControl::FirmwareUpgrade, 99
- shutDownReasonCodes
 - CrossControl, 19
- ShutdownPending
 - CrossControl, 19
- shutdownReasonCodeNoError
 - CrossControl, 20
- Smart, 138
- Smart.h
 - GetSmart, 197
 - SMARTHANDLE, 197
- startAutomaticBL
 - CrossControl::Backlight, 53
- startAverageCalc
 - CrossControl::Lightsensor, 119
- startFpgaUpgrade
 - CrossControl::FirmwareUpgrade, 99
- startFpgaVerification
 - CrossControl::FirmwareUpgrade, 100
- startFrontUpgrade
 - CrossControl::FirmwareUpgrade, 101
- startFrontVerification
 - CrossControl::FirmwareUpgrade, 102
- startSSUpgrade
 - CrossControl::FirmwareUpgrade, 103
- startSSVerification
 - CrossControl::FirmwareUpgrade, 104
- startupReasonCodeButtonPress
 - CrossControl, 20
- startupReasonCodeExtCtrl
 - CrossControl, 20
- startupReasonCodeMPRestart
 - CrossControl, 20
- startupReasonCodePowerOnStartup
 - CrossControl, 20
- startupReasonCodeUndefined
 - CrossControl, 20
- startupReasonCodes
 - CrossControl, 20
- stopAutomaticBL
 - CrossControl::Backlight, 53
- stopAverageCalc
 - CrossControl::Lightsensor, 119
- SuspendPending
 - CrossControl, 19
- TEMP_BACKPLANE
 - CrossControl, 18
- TEMP_BOARD
 - CrossControl, 18
- TEMP_BOX
 - CrossControl, 18
- TEMP_CHIPSETS
 - CrossControl, 18
- TEMP_CPU
 - CrossControl, 18
- TEMP_ENV
 - CrossControl, 18
- TEMP_OTHER
 - CrossControl, 18
- TEMP_VIDEO
 - CrossControl, 18
- TOUCH_NEXT_BOOT
 - CrossControl, 20
- TOUCH_NOW
 - CrossControl, 20
- TS_CALIBRATION_MEASUREMENT-
S
 - CrossControl, 21
- TS_CALIBTATION_WIDTH
 - CrossControl, 21
- TS_DEBOUNCE_TIME
 - CrossControl, 21
- TS_DEBOUNCE_TIMEOUT_TIME
 - CrossControl, 21
- TS_DOUBLE_CLICK_TIME
 - CrossControl, 21
- TS_DOUBLECLICK_MAX_CLICK_TI-
ME
 - CrossControl, 21
- TS_LOW_LEVEL
 - CrossControl, 21
- TS_MAX_RIGHTCLICK_DISTANCE
 - CrossControl, 21
- TS_RESTORE_DEFAULT_SETTINGS
 - CrossControl, 21
- TS_RIGHT_CLICK_TIME
 - CrossControl, 21
- TS_UNTOUCHLEVEL
 - CrossControl, 21
- TS_USE_DEJITTER
 - CrossControl, 21
- TELEMATICSHANDLE

- Telematics.h, 198
- TOUCHSCREENHANDLE
 - TouchScreen.h, 200
- TSAAdvancedSettingsParameter
 - CrossControl, 20
- takeSnapshot
 - CrossControl::Video, 173
- takeSnapshotBmp
 - CrossControl::Video, 173
- takeSnapshotRaw
 - CrossControl::Video, 174
- Telematics, 141
- Telematics.h
 - GetTelematics, 198
 - TELEMATICSHANDLE, 198
- TimerType, 153
- TotHeatTime
 - CrossControl::TimerType, 153
- TotRunTime
 - CrossControl::TimerType, 154
- TotRunTimeBattery
 - CrossControl::BatteryTimerType, 66
- TotRunTimeMain
 - CrossControl::BatteryTimerType, 66
- TotSuspTime
 - CrossControl::TimerType, 154
- TouchScreen, 154
- TouchScreen.h
 - GetTouchScreen, 200
 - TOUCHSCREENHANDLE, 200
- TouchScreenCalib, 158
- TouchScreenCalib.h
 - GetTouchScreenCalib, 201
- TouchScreenModeSettings
 - CrossControl, 20
- TriggerConf
 - CrossControl, 20
- UNDEFINED_COLOR
 - CrossControl, 12
- UPGRADE_COMPLETE
 - CrossControl, 22
- UPGRADE_COMPLETE_WITH_ERRORS
 - CrossControl, 22
- UPGRADE_CONVERTING_FILE
 - CrossControl, 21
- UPGRADE_FLASHING
 - CrossControl, 22
- UPGRADE_INIT
 - CrossControl, 21
- UPGRADE_PREP_COM
 - CrossControl, 21
- UPGRADE_READING_FILE
 - CrossControl, 21
- UPGRADE_VERIFYING
 - CrossControl, 22
- UpgradeAction
 - CrossControl, 21
- UpgradeStatus, 161
- VOLTAGE_0V9
 - CrossControl, 23
- VOLTAGE_12V
 - CrossControl, 22
- VOLTAGE_12VID
 - CrossControl, 22
- VOLTAGE_1V0
 - CrossControl, 23
- VOLTAGE_1V05
 - CrossControl, 23
- VOLTAGE_1V2
 - CrossControl, 23
- VOLTAGE_1V5
 - CrossControl, 23
- VOLTAGE_1V8
 - CrossControl, 23
- VOLTAGE_1V9
 - CrossControl, 23
- VOLTAGE_24V
 - CrossControl, 22
- VOLTAGE_24VIN
 - CrossControl, 22
- VOLTAGE_3V3
 - CrossControl, 22
- VOLTAGE_5V
 - CrossControl, 22
- VOLTAGE_5VSTB
 - CrossControl, 22
- VOLTAGE_VREF_INT
 - CrossControl, 23
- VOLTAGE_VTFT
 - CrossControl, 22
- VIDEOHANDLE
 - Video.h, 202
- version_info, 162
- VersionType
 - CrossControl, 10
- Video, 162
- Video.h

- GetVideo, [202](#)
- VIDEOHANDLE, [202](#)
- Video1Conf
 - CrossControl, [25](#)
- Video2Conf
 - CrossControl, [25](#)
- Video3Conf
 - CrossControl, [25](#)
- Video4Conf
 - CrossControl, [25](#)
- video_dec_command, [174](#)
- VideoChannel
 - CrossControl, [22](#)
- videoStandard
 - CrossControl, [22](#)
- VoltageEnum
 - CrossControl, [22](#)
- volume
 - CrossControl::BuzzerSetup, [72](#)
- YELLOW
 - CrossControl, [12](#)