

**crosscontrol**

**CCAux**

2.17.6.0

Jan 20 2021

# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	Introduction	1
1.2	Changelog	1
1.2.1	Version 2.17.7.0 - next release	1
1.2.2	Version 2.17.6.0 - V700 platform	1
1.2.3	Version 2.17.5.0 - XL5, V700 platforms	2
1.2.4	Version 2.17.3.0 - XL5 platform	2
1.2.5	Version 2.17.2.0 - XM9 platform	2
1.2.6	Version 2.17.1.0 - XM9-Win platform	2
1.2.7	Version 2.17.0.0 - XM9, VS, VI2 platforms	2
1.2.8	Version 2.16.2.0 - XM2, XM9, VS, VI2 platforms	3
1.2.9	Version 2.16.1.0 - XM2 platform	3
1.2.10	Version 2.16.0.0 - VS and VI2 platforms	3
1.2.11	Version 2.15.1.0 - VC platform	4
1.2.12	Version 2.15.0.0 - VI2 platform	4
1.2.13	Version 2.14.0.23 - XM9 platform	4
1.2.14	Version 2.14.0.0 - all platforms	4
1.2.15	Version 2.13.2.25 - XM9 platform	4
1.2.16	Version 2.13.0.0 - VS and VI2 platforms	4
1.2.17	Version 2.12.0.0 - VS platform	5
1.2.18	Version 2.11.0.0 - VS platform	5
1.2.19	Version 2.10.0.0 - VS platform	5
1.2.20	Version 2.9.0.0 - XA/XS/VC/VA and XM2 Linux platforms	5
1.2.21	Version 2.8.3.0 - VC/VA Linux platforms	5
1.2.22	Version 2.7.4.0 - VC Linux platform	6
1.2.23	Version 2.7.3.0 - VC Linux platform	6
1.2.24	Version 2.7.2.0 - XM/XL Windows x86, x64 platform, VC Linux platform	6
1.2.25	Version 2.7.1.0 - XM/XL Windows x86, x64 platform, VC Linux platform	6
1.2.26	Version 2.7.0.0 - XM/XL Windows x86, x64 platform, VC Linux platform	6
1.2.27	Version 2.6.2.0 - XM/XL Windows x86, x64 platform	7
1.2.28	Version 2.6.1.0 - XM/XL Windows x86, x64 platform and XA/XS Linux platform	7
1.2.29	Version 2.5.0.0 - XM/XL x86 platform	8
1.2.30	Version 2.4.7.0 - XM Linux platform	9
1.2.31	Version 2.4.6.0 - XA/XS platform	9

1.2.32	Version 2.4.2.0 - XA/XS platform	9
1.2.33	Version 2.4.0.0 - XA/XS, XM platforms	9
1.2.34	Version 2.3.0.0 - XA/XS platform	11
1.2.35	Version 2.2.0.0 - XA/XS platform	11
1.2.36	Version 2.1.0.0 - XA/XS platform	12
1.2.37	Version 2.0.0.0 - XA/XS platform	12
1.3	Known Issues	13
<b>2</b>	<b>Functions added in v2.17.0.0</b>	<b>14</b>
<b>3</b>	<b>Functions added in v2.16.2.0</b>	<b>15</b>
<b>4</b>	<b>Functions added in v2.15.0.0</b>	<b>16</b>
<b>5</b>	<b>Functions added in v2.14.0.0</b>	<b>17</b>
<b>6</b>	<b>Functions added in v2.13.0.0</b>	<b>18</b>
<b>7</b>	<b>Functions added in v2.12.0.0</b>	<b>19</b>
<b>8</b>	<b>Functions added in v2.11.0.0</b>	<b>20</b>
<b>9</b>	<b>Functions added in v2.10.0.0</b>	<b>21</b>
<b>10</b>	<b>Functions added in v2.8.3.0</b>	<b>22</b>
<b>11</b>	<b>Functions added in v2.7.3.0</b>	<b>23</b>
<b>12</b>	<b>Functions added in v2.7.0.0</b>	<b>24</b>
<b>13</b>	<b>Functions added in v2.6.1.0</b>	<b>25</b>
<b>14</b>	<b>Functions added in v2.5.0.0</b>	<b>26</b>
<b>15</b>	<b>Module Index</b>	<b>27</b>
15.1	Modules	27
<b>16</b>	<b>Namespace Index</b>	<b>28</b>
16.1	Namespace List	28
<b>17</b>	<b>Data Structure Index</b>	<b>29</b>
17.1	Data Structures	29
<b>18</b>	<b>File Index</b>	<b>30</b>
18.1	File List	30
<b>19</b>	<b>Module Documentation</b>	<b>31</b>
19.1	About functions	31
19.1.1	Detailed Description	33
19.1.2	Function Documentation	33
	About_getAddOnHWversion()	33
	About_getAddOnManufacturingDate()	33
	About_getAddOnPCBArt()	34

About_getAddOnPCBSerial()	34
About_getDisplayResolution()	35
About_getFrontPcbRev()	35
About_getIOExpanderValue()	36
About_getIsAnybusMounted()	36
About_getIsBTMounted()	37
About_getIsDisplayAvailable()	37
About_getIsGPRSMounted()	37
About_getIsGPSPMounted()	38
About_getIsIOExpanderMounted()	38
About_getIsPCBBluetoothAvailable()	40
About_getIsTouchScreenAvailable()	40
About_getIsWLANMounted()	41
About_getMainHWversion()	41
About_getMainManufacturingDate()	42
About_getMainPCBArt()	42
About_getMainPCBSerial()	43
About_getMainProdArtNr()	43
About_getMainProdRev()	44
About_getNrOfAnalogInputs()	45
About_getNrOfButtons()	45
About_getNrOfCANConnections()	45
About_getNrOfCfgInConnections()	46
About_getNrOfDigIOConnections()	46
About_getNrOfETHConnections()	47
About_getNrOfPWMOutConnections()	47
About_getNrOfSerialConnections()	48
About_getNrOfUSBConnections()	48
About_getNrOfVideoConnections()	49
About_getUnitSerial()	49
About_getUserEepromData()	50
About_hasOsBooted()	50
About_release()	51
About_setUserEepromData()	51
GetAbout()	52
19.2 Adc functions	52
19.2.1 Detailed Description	52
19.2.2 Function Documentation	52
Adc_getVoltage()	52
Adc_release()	54
GetAdc()	55
19.3 AuxVersion functions	56
19.3.1 Detailed Description	56
19.3.2 Function Documentation	56
AuxVersion_getCanFdApiVersion()	57
AuxVersion_getCanFdMcuVersion()	57
AuxVersion_getCCAuxDrvVersion()	58
AuxVersion_getCCAuxVersion()	59
AuxVersion_getFPGAVersion()	59
AuxVersion_getFrontVersion()	60

AuxVersion_getFullCCAuxVersion()	61
AuxVersion_getFullOSVersion()	62
AuxVersion_getOSVersion()	62
AuxVersion_getSSVersion()	63
AuxVersion_release()	64
GetAuxVersion()	64
19.4 Backlight functions	65
19.4.1 Detailed Description	65
19.4.2 Function Documentation	65
Backlight_getAutomaticBLFilter()	66
Backlight_getAutomaticBLParams()	66
Backlight_getAutomaticBLStatus()	67
Backlight_getHWStatus()	67
Backlight_getIntensity()	68
Backlight_getLedDimming()	68
Backlight_getStatus()	69
Backlight_release()	69
Backlight_setAutomaticBLFilter()	70
Backlight_setAutomaticBLParams()	70
Backlight_setIntensity()	71
Backlight_setLedDimming()	71
Backlight_startAutomaticBL()	72
Backlight_stopAutomaticBL()	72
GetBacklight()	73
19.5 Battery functions	73
19.5.1 Detailed Description	74
19.5.2 Function Documentation	74
Battery_getBatteryChargingStatus()	74
Battery_getBatteryHWversion()	75
Battery_getBatterySerial()	75
Battery_getBatterySwVersion()	76
Battery_getBatteryTemp()	77
Battery_getBatteryVoltageStatus()	78
Battery_getHwErrorStatus()	78
Battery_getMinMaxTemp()	79
Battery_getPowerSource()	80
Battery_getTimer()	81
Battery_isBatteryPresent()	81
Battery_release()	82
GetBattery()	82
19.6 Buzzer functions	83
19.6.1 Detailed Description	83
19.6.2 Function Documentation	83
Buzzer_buzze()	84
Buzzer_getFrequency()	84
Buzzer_getScaledVolume()	85
Buzzer_getTrigger()	85
Buzzer_getVolume()	85
Buzzer_release()	86
Buzzer_setFrequency()	86

	Buzzer_setScaledVolume()	87
	Buzzer_setTrigger()	87
	Buzzer_setVolume()	88
	GetBuzzer()	88
19.7	CanSetting functions	89
19.7.1	Detailed Description	89
19.7.2	Function Documentation	89
	CanSetting_getBaudrate()	89
	CanSetting_getFlexibleDataBaudrates()	90
	CanSetting_getFrameType()	90
	CanSetting_release()	91
	CanSetting_setBaudrate()	91
	CanSetting_setFlexibleDataBaudrates()	92
	CanSetting_setFrameType()	92
	GetCanSetting()	93
19.8	CfgIn functions	93
19.8.1	Detailed Description	94
19.8.2	Function Documentation	94
	CfgIn_getCfgInMode()	94
	CfgIn_getFrequencyValue()	95
	CfgIn_getMinFrequencyThreshold()	96
	CfgIn_getPwmValue()	97
	CfgIn_getValue()	97
	CfgIn_release()	98
	CfgIn_setCfgInMode()	99
	CfgIn_setFrequencyFilterLevel()	99
	CfgIn_setMinFrequencyThreshold()	100
	GetCfgIn()	101
19.9	Config functions	101
19.9.1	Detailed Description	103
19.9.2	Function Documentation	103
	Config_getButtonFunction()	103
	Config_getCanStartupPowerConfig()	104
	Config_getDigPowerOutputStartupConfig()	105
	Config_getExtFanStartupPowerConfig()	105
	Config_getExtOnOffSigTrigTime()	106
	Config_getFrontBtnTrigTime()	106
	Config_getHeatingTempLimit()	106
	Config_getLongBeepSettings()	107
	Config_getLongButtonPressAction()	107
	Config_getNextBootMode()	108
	Config_getOnOffSigAction()	108
	Config_getOnOffSignalState()	109
	Config_getOnOffTriggerMode()	109
	Config_getOSAliveMonitoring()	109
	Config_getPowerOnStartup()	110
	Config_getRS485Enabled()	110
	Config_getShortBeepSettings()	111
	Config_getShortButtonPressAction()	111
	Config_getStartupTriggerConfig()	112

Config_getStartupVoltageConfig() . . . . .	113
Config_getSuspendMaxTime() . . . . .	113
Config_getVideoStartupPowerConfig() . . . . .	114
Config_release() . . . . .	114
Config_setButtonFunction() . . . . .	114
Config_setCanStartupPowerConfig() . . . . .	116
Config_setDigPowerOutputStartupConfig() . . . . .	116
Config_setExtFanStartupPowerConfig() . . . . .	117
Config_setExtOnOffSigTrigTime() . . . . .	117
Config_setFrontBtnTrigTime() . . . . .	118
Config_setHeatingTempLimit() . . . . .	118
Config_setLongBeepSettings() . . . . .	118
Config_setLongButtonPressAction() . . . . .	119
Config_setNextBootMode() . . . . .	119
Config_setOnOffSigAction() . . . . .	120
Config_setOnOffTriggerMode() . . . . .	120
Config_setOSAliveMonitoring() . . . . .	121
Config_setPowerOnStartup() . . . . .	121
Config_setRS485Enabled() . . . . .	122
Config_setShortBeepSettings() . . . . .	122
Config_setShortButtonPressAction() . . . . .	123
Config_setStartupTriggerConfig() . . . . .	123
Config_setStartupVoltageConfig() . . . . .	124
Config_setSuspendMaxTime() . . . . .	124
Config_setVideoStartupPowerConfig() . . . . .	125
GetConfig() . . . . .	125
19.10 Diagnostic functions . . . . .	125
19.10.1 Detailed Description . . . . .	126
19.10.2 Function Documentation . . . . .	126
Diagnostic_clearHwErrorStatus() . . . . .	126
Diagnostic_getHwErrorStatus() . . . . .	126
Diagnostic_getMinMaxTemp() . . . . .	127
Diagnostic_getPCBTemp() . . . . .	127
Diagnostic_getPMTemp() . . . . .	128
Diagnostic_getPowerCycles() . . . . .	128
Diagnostic_getShutDownReason() . . . . .	129
Diagnostic_getSSTemp() . . . . .	129
Diagnostic_getStartupReason() . . . . .	130
Diagnostic_getTimer() . . . . .	130
Diagnostic_release() . . . . .	131
GetDiagnostic() . . . . .	131
19.11 DigIO functions . . . . .	131
19.11.1 Detailed Description . . . . .	132
19.11.2 Function Documentation . . . . .	132
DigIO_getDigIO() . . . . .	132
DigIO_getDigPowerOutput() . . . . .	132
DigIO_release() . . . . .	133
DigIO_setDigIO() . . . . .	134
DigIO_setDigPowerOutput() . . . . .	134
GetDigIO() . . . . .	135

19.12 FirmwareUpgrade functions	135
19.12.1 Detailed Description	136
19.12.2 Function Documentation	136
FirmwareUpgrade_getUpgradeStatus()	136
FirmwareUpgrade_release()	136
FirmwareUpgrade_shutDown()	137
FirmwareUpgrade_startDMCUUpgrade()	137
FirmwareUpgrade_startDMCUVerification()	138
FirmwareUpgrade_startFpgaUpgrade()	138
FirmwareUpgrade_startFpgaVerification()	139
FirmwareUpgrade_startFrontUpgrade()	140
FirmwareUpgrade_startFrontVerification()	141
FirmwareUpgrade_startSSUpgrade()	142
FirmwareUpgrade_startSSVerification()	143
GetFirmwareUpgrade()	144
19.13 FrontLED functions	145
19.13.1 Detailed Description	146
19.13.2 Function Documentation	146
FrontLED_getBootLEDConfig()	146
FrontLED_getColor()	147
FrontLED_getEnabledDuringStartup()	147
FrontLED_getIdleTime()	148
FrontLED_getNrOfPulses()	148
FrontLED_getOffTime()	148
FrontLED_getOnTime()	149
FrontLED_getPostBootLEDConfig()	149
FrontLED_getSignal()	150
FrontLED_getStandardColor()	151
FrontLED_release()	151
FrontLED_setBootLEDConfig()	151
FrontLED_setColor()	152
FrontLED_setEnabledDuringStartup()	152
FrontLED_setIdleTime()	153
FrontLED_setNrOfPulses()	153
FrontLED_setOff()	154
FrontLED_setOffTime()	154
FrontLED_setOnTime()	154
FrontLED_setPostBootLEDConfig()	155
FrontLED_setSignal()	156
FrontLED_setStandardColor()	156
GetFrontLED()	157
19.14 Lightsensor functions	157
19.14.1 Detailed Description	158
19.14.2 Function Documentation	158
GetLightsensor()	158
Lightsensor_getAverageIlluminance()	158
Lightsensor_getIlluminance()	159
Lightsensor_getIlluminance2()	159
Lightsensor_getOperatingRange()	160
Lightsensor_release()	160



Lightsensor_setOperatingRange()	161
Lightsensor_startAverageCalc()	161
Lightsensor_stopAverageCalc()	162
19.15 Power functions	162
19.15.1 Detailed Description	163
19.15.2 Function Documentation	163
GetPower()	163
Power_ackPowerRequest()	164
Power_getBLPowerStatus()	164
Power_getButtonPowerTransitionStatus()	165
Power_getCanOCDSStatus()	165
Power_getCanPowerStatus()	166
Power_getExtFanPowerStatus()	166
Power_getVideoOCDSStatus()	167
Power_getVideoPowerStatus()	167
Power_release()	168
Power_setBLPowerStatus()	168
Power_setCanPowerStatus()	169
Power_setExtFanPowerStatus()	169
Power_setVideoPowerStatus()	170
19.16 PowerMgr functions	170
19.16.1 Detailed Description	171
19.16.2 Function Documentation	171
GetPowerMgr()	171
PowerMgr_getConfiguration()	171
PowerMgr_getPowerMgrStatus()	172
PowerMgr_hasResumed()	174
PowerMgr_registerControlledSuspendOrShutDown()	176
PowerMgr_release()	177
PowerMgr_setAppReadyForSuspendOrShutdown()	177
19.17 PWMOut functions	179
19.17.1 Detailed Description	180
19.17.2 Function Documentation	180
GetPWMOut()	180
PWMOut_getPWMOutputChannelDutyCycle()	180
PWMOut_getPWMOutputChannelFrequency()	181
PWMOut_getPWMOutputStatus()	181
PWMOut_release()	182
PWMOut_setPWMOutOff()	183
PWMOut_setPWMOutputChannelDutyCycle()	183
PWMOut_setPWMOutputChannelFrequency()	184
19.18 Smart functions	184
19.18.1 Detailed Description	185
19.18.2 Function Documentation	185
GetSmart()	185
Smart_getDeviceSerial()	185
Smart_getDeviceSerial2()	187
Smart_getInitialTime()	188
Smart_getInitialTime2()	189
Smart_getRemainingLifeTime()	189

Smart_getRemainingLifeTime2()	190
Smart_release()	191
19.19 SoftKey functions	191
19.19.1 Detailed Description	192
19.19.2 Function Documentation	192
GetSoftKey()	192
SoftKey_getBacklightEnabledDuringStartup()	193
SoftKey_getBacklightIdleTime()	193
SoftKey_getBacklightIntensity()	194
SoftKey_getBacklightNrOfPulses()	194
SoftKey_getBacklightOffTime()	195
SoftKey_getBacklightOnTime()	195
SoftKey_getBacklightSignal()	195
SoftKey_getBootBacklightConfig()	196
SoftKey_getMultipleBacklightIntensities()	197
SoftKey_getStatus()	197
SoftKey_release()	199
SoftKey_setBacklightEnabledDuringStartup()	199
SoftKey_setBacklightIdleTime()	200
SoftKey_setBacklightIntensity()	200
SoftKey_setBacklightNrOfPulses()	201
SoftKey_setBacklightOff()	201
SoftKey_setBacklightOffTime()	201
SoftKey_setBacklightOnTime()	202
SoftKey_setBacklightSignal()	202
SoftKey_setBootBacklightConfig()	203
SoftKey_setMultipleBacklightIntensities()	204
19.20 Telematics functions	205
19.20.1 Detailed Description	206
19.20.2 Function Documentation	206
GetTelematics()	206
Telematics_getBTPowerStatus()	206
Telematics_getBTStartUpPowerStatus()	207
Telematics_getGPRSPowerStatus()	207
Telematics_getGPRSStartUpPowerStatus()	208
Telematics_getGPSAntennaStatus()	209
Telematics_getGPSPowerStatus()	209
Telematics_getGPSStartUpPowerStatus()	210
Telematics_getTelematicsAvailable()	211
Telematics_getWLANPowerStatus()	211
Telematics_getWLANStartUpPowerStatus()	212
Telematics_release()	213
Telematics_setBTPowerStatus()	213
Telematics_setBTStartUpPowerStatus()	213
Telematics_setGPRSPowerStatus()	214
Telematics_setGPRSStartUpPowerStatus()	214
Telematics_setGPSPowerStatus()	214
Telematics_setGPSStartUpPowerStatus()	215
Telematics_setWLANPowerStatus()	215
Telematics_setWLANStartUpPowerStatus()	216

19.21 TouchScreen functions	216
19.21.1 Detailed Description	217
19.21.2 Function Documentation	217
GetTouchScreen()	217
TouchScreen_getAdvancedSetting()	217
TouchScreen_getMode()	218
TouchScreen_getMouseRightClickTime()	218
TouchScreen_release()	219
TouchScreen_setAdvancedSetting()	219
TouchScreen_setMode()	221
TouchScreen_setMouseRightClickTime()	221
19.22 TouchScreenCalib functions	222
19.22.1 Detailed Description	222
19.22.2 Function Documentation	222
GetTouchScreenCalib()	222
TouchScreenCalib_autoSensorCalib()	223
TouchScreenCalib_checkCalibrationPointFinished()	223
TouchScreenCalib_getConfigParam()	223
TouchScreenCalib_getMode()	224
TouchScreenCalib_release()	224
TouchScreenCalib_setCalibrationPoint()	224
TouchScreenCalib_setConfigParam()	225
TouchScreenCalib_setMode()	225
19.23 Video functions	226
19.23.1 Detailed Description	227
19.23.2 Function Documentation	227
GetVideo()	228
Video_activateSnapshot()	228
Video_createBitmap()	228
Video_freeBmpBuffer()	229
Video_getActiveChannel()	229
Video_getColorKeys()	230
Video_getCropping()	230
Video_getDecoderReg()	231
Video_getDeInterlaceMode()	231
Video_getGraphicsOverlay()	232
Video_getMirroring()	232
Video_getRawImage()	233
Video_getRotation()	233
Video_getScaling()	234
Video_getStatus()	234
Video_getVideoArea()	235
Video_getVideoStandard()	235
Video_init()	236
Video_minimize()	236
Video_release()	236
Video_restore()	237
Video_setActiveChannel()	237
Video_setColorKeys()	237
Video setCropping()	238

Video.setDecoderReg()	238
Video.setDeInterlaceMode()	239
Video.setGraphicsOverlay()	239
Video.setMirroring()	240
Video.setRotation()	240
Video.setScaling()	241
Video.setVideoArea()	241
Video.showFrame()	242
Video.showVideo()	242
Video.takeSnapshot()	242
Video.takeSnapshotBmp()	243
Video.takeSnapshotRaw()	244
<b>20 Namespace Documentation</b>	<b>245</b>
20.1 CrossControl Namespace Reference	245
20.1.1 Typedef Documentation	265
ABOUTHANDLE	265
ADCHANDLE	265
AUXVERSIONHANDLE	265
BACKLIGHTHANDLE	266
BATTERYHANDLE	266
BUZZERHANDLE	266
CANSETTINGHANDLE	266
CFGINHANDLE	266
CONFIGHANDLE	266
DIAGNOSTICHANDLE	266
DIGIOHANDLE	266
FIRMWAREUPGHANDLE	266
FRONTLEDHANDLE	266
LIGHTSENSORHANDLE	267
POWERHANDLE	267
POWERMGRHANDLE	267
PWMOUTHANDLE	267
SMARTHANDLE	267
SOFTKEYHANDLE	267
TELEMATICSHANDLE	267
TOUCHSCREENCALIBHANDLE	267
TOUCHSCREENHANDLE	267
VersionType	267
VIDEOHANDLE	267
20.1.2 Enumeration Type Documentation	268
BootModeEnum	268
ButtonConfigEnum	268
ButtonPowerTransitionStatus	269
CalibrationConfigParam	269
CalibrationModeSettings	270
CanFrameType	270
CanMode	270
CCAuxColor	271
CCStatus	271

CfgInModeEnum	271
ChargingStatus	273
ConfigOnOffTriggerMode	274
DeInterlaceMode	274
eErr	274
ErrorStatus	276
hwErrorStatusCodes	276
JidaSensorType	277
LightSensorOperationRange	277
LightSensorSamplingMode	277
OCDStatus	278
PowerAction	278
PowerMgrConf	278
PowerMgrStatus	279
PowerOutput	279
PowerSource	279
RS4XXPort	279
shutDownReasonCodes	280
startupReasonCodes	280
SystemMode	281
TouchScreenModeSettings	281
TriggerConf	281
TSAdvancedSettingsParameter	282
UpgradeAction	284
VideoChannel	284
VideoRotation	285
videoStandard	285
VoltageEnum	285
20.1.3 Function Documentation	287
GetErrorStringA()	287
GetErrorStringW()	288
GetHwErrorStatusStringA()	288
GetHwErrorStatusStringW()	288
GetStartupReasonStringA()	289
GetStartupReasonStringW()	289
20.1.4 Variable Documentation	289
DigitalIn_1	290
DigitalIn_2	290
DigitalIn_3	290
DigitalIn_4	290
Video1Conf	290
Video2Conf	290
Video3Conf	290
Video4Conf	290

<b>21 Data Structure Documentation</b>	<b>291</b>
21.1 BatteryTimerType Struct Reference	291
21.1.1 Field Documentation	291
RunTime_0_40	291
RunTime_40_60	291
RunTime_60_70	291
RunTime_70_80	292
RunTime_Above80	292
RunTime_m20	292
RunTime_m20_0	292
TotRunTimeBattery	292
TotRunTimeMain	292
21.2 BuzzerSetup Struct Reference	292
21.2.1 Field Documentation	292
frequency	292
volume	293
21.3 FpgaLedTimingType Struct Reference	293
21.3.1 Field Documentation	293
idleTime	293
ledNbr	293
nrOfPulses	293
offTime	293
onTime	293
21.4 LedColorMixType Struct Reference	294
21.4.1 Field Documentation	294
blue	294
green	294
red	294
21.5 LedTimingType Struct Reference	294
21.5.1 Field Documentation	294
idleTime	294
nrOfPulses	295
offTime	295
onTime	295
21.6 received_video Struct Reference	295
21.6.1 Field Documentation	295
received_framerate	295
received_height	295
received_width	295
21.7 TimerType Struct Reference	296
21.7.1 Detailed Description	296
21.7.2 Field Documentation	296
Above80RunTime	296
RunTime40_60	296
RunTime60_70	296
RunTime70_80	296
TotHeatTime	296
TotRunTime	296
TotSuspTime	297
21.8 UpgradeStatus Struct Reference	297

21.8.1	Detailed Description	297
21.8.2	Field Documentation	297
	currentAction	297
	errorCode	297
	percent	297
21.9	version_info Struct Reference	297
21.9.1	Field Documentation	298
	build	298
	major	298
	minor	298
	release	298
21.10	video_dec_command Struct Reference	298
21.10.1	Field Documentation	298
	decoder_register	298
	register_value	299
<b>22</b>	<b>File Documentation</b>	<b>300</b>
22.1	IncludeFiles/About.h File Reference	300
22.2	IncludeFiles/Adc.h File Reference	302
22.3	IncludeFiles/AuxVersion.h File Reference	302
22.4	IncludeFiles/Backlight.h File Reference	303
22.5	IncludeFiles/Battery.h File Reference	304
22.6	IncludeFiles/Buzzer.h File Reference	305
22.7	IncludeFiles/CanSetting.h File Reference	306
22.8	IncludeFiles/CCAuxErrors.h File Reference	307
22.9	IncludeFiles/CCAuxTypes.h File Reference	307
22.9.1	Typedef Documentation	310
	char_t	310
	float32_t	310
	float64_t	310
	int16_t	311
	int32_t	311
	int64_t	311
	int8_t	311
	uint16_t	311
	uint32_t	311
	uint64_t	311
	uint8_t	311
22.10	IncludeFiles/CCPlatform.h File Reference	311
22.11	IncludeFiles/CfgIn.h File Reference	311
22.12	IncludeFiles/Config.h File Reference	312
22.13	IncludeFiles/Diagnostic.h File Reference	314
22.14	IncludeFiles/DiagnosticCodes.h File Reference	315
22.15	IncludeFiles/DigIO.h File Reference	316
22.16	IncludeFiles/FirmwareUpgrade.h File Reference	316
22.17	IncludeFiles/FrontLED.h File Reference	317
22.18	IncludeFiles/Lightsensor.h File Reference	319
22.19	IncludeFiles/Power.h File Reference	319
22.20	IncludeFiles/PowerMgr.h File Reference	320
22.21	IncludeFiles/PWMOut.h File Reference	321

*CONTENTS*

xv

22.22IncludeFiles/Releasenotes.dox File Reference . . . . .	321
22.23IncludeFiles/Smart.h File Reference . . . . .	321
22.24IncludeFiles/SoftKey.h File Reference . . . . .	322
22.25IncludeFiles/Telematics.h File Reference . . . . .	323
22.26IncludeFiles/TouchScreen.h File Reference . . . . .	324
22.27IncludeFiles/TouchScreenCalib.h File Reference . . . . .	325
22.28IncludeFiles/Video.h File Reference . . . . .	326

**Index**

**329**



# 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.

The API is available for multiple platforms and operating systems: Linux on the CCpilot XA, XS, VC, VA, XM, VI2, VS, X900 and V700 products in all variations. For the XM and XL platforms, Windows XP, Windows 7, 8 and 10 is also supported.

The known issues and changelog presented here also cover the following [CrossControl](#) applications (which are using the API and are released in conjunction with it):

- ccsettings
- ccvideo
- ccsettingsconsole
- touchcalibrator
- ccauxd

Go directly to the [complete list of functions](#) or the [functions by class](#).

### 1.2 Changelog

#### 1.2.1 Version 2.17.7.0 - next release

- New feature: Added initial support for new platform. (Yukon)

#### 1.2.2 Version 2.17.6.0 - V700 platform

- Change: Removed CAN\_POWER and invalid backlight control (V700)

- ccsettingsconsole: Removed unsupported features. (V700)
- ccsettingsconsole: Bugfix: `--about=RawEEPROM` prints segfault. (V700)
- ccsettingsconsole: New feature: CPU temp can now be read. (V700)
- ccsettingsconsole: New feature: Added support for setting minimum automatic backlight intensity. (ALL)
- ccsettingsconsole: New feature: Added support for beep settings. (V700, VS)
- ccsettingsconsole: New feature: Added support for setting boot and post-boot LED settings. (V700, VS, VI2)
- ccsettingsconsole: Bugfix: Wrong order for ADC voltage strings (V700)

### 1.2.3 Version 2.17.5.0 - XL5, V700 platforms

- Backlight parameters settable from registry (CCAuxService - XM9-Win, XL5-Win)
- Initial release for v700 platform

### 1.2.4 Version 2.17.3.0 - XL5 platform

- Bugfix for `Video_getDecoderReg` which was broken in 2.9.0.0 (XM, XL)
- Initial release for XL5 platform

### 1.2.5 Version 2.17.2.0 - XM9 platform

- Removed telematics support (XM9)
- Removed suspend support (XM9-Windows)

### 1.2.6 Version 2.17.1.0 - XM9-Win platform

- Support for SS firmware update over SPI (XM9-Win)
- Now supporting the `Config_getOnOffSignalState` function (XM9)

### 1.2.7 Version 2.17.0.0 - XM9, VS, VI2 platforms

- Suspend support added - VS platform
- NTSC video support added - VS platform
- Bugfix in several Backlight functions. No longer allow automatic BL for devices without light sensor.
- Bugfix for `About_getUnitSerial()` - VS platform
- Bugfix: `Config_set/getOnOffTriggerMode` now returns `ERR_NOT_SUPPORTED` when used with SS older than 1.1.0.0 as using `Config_setOnOffTriggerMode` could make the device not able to boot with older SS versions - VS platform

- New function [AuxVersion.getCanFdApiVersion\(\)](#) - XM9 platform
- New function [AuxVersion.getFullOSVersion\(\)](#)
- New function [AuxVersion.getFullCCAuxVersion\(\)](#)
- ccsettingsconsole: Added error message if user tries to update SS from normal system - (VS & VI2)
- ccsettingsconsole: Added support to get/set CAN-FD data baud rate - (XM9)
- ccsettingsconsole: System Report added (ccsettingsconsole --about=SystemReport)
- ccsettings: Added support to get/set CAN-FD data baud rate (XM9)
- ccsettings: Tag names of CCAux and OS versions are now shown in version tab and in the system report
- ccsettings: Added CAN-FD MCU to version tab (XM9)
- ccsettings: Added CAN-FD MCU SW update & verification (XM9)

### 1.2.8 Version 2.16.2.0 - XM2, XM9, VS, VI2 platforms

- CCPilot XM9 support.
- Added the following functions:
  - [CanSetting.getFlexibleDataBaudrates\(\)](#)
  - [CanSetting.setFlexibleDataBaudrates\(\)](#)
  - [AuxVersion.getCanFdMcuVersion\(\)](#)

### 1.2.9 Version 2.16.1.0 - XM2 platform

- Bugfix for supported platforms of [About.getNrOfPWMOutConnections\(\)](#).
- Bugfix for supported platforms of the `Telematic` class functions.
- Bugfix for [Video.getDecoderReg\(\)](#) function. Fixes bug introduced in 2.9.0.0 where this function stopped working.

### 1.2.10 Version 2.16.0.0 - VS and VI2 platforms

- Bugfix for ICM EEPROM read/writes - VS and VI2 platforms
- Updated description of how to use error status byte in [DigIO.getDigPowerOutput\(\)](#) - VS platform
- Added support for [About.getNrOfCfgInConnections\(\)](#) - VS platform
- Bugfix for [About.getNrOfSerialConnections\(\)](#) - VS platform

### 1.2.11 Version 2.15.1.0 - VC platform

- Bugfix for a multithreading issue affecting the following classes:
  - [About](#)
  - [Auxversion](#)
  - [Backlight](#)
  - [Battery](#)
  - [Buzzer](#)
  - [CfgIn](#)
  - [Config](#)
  - [Diagnostic](#)
  - [FrontLED](#)
  - [Telematics](#)

### 1.2.12 Version 2.15.0.0 - VI2 platform

- New function [About.getNrOfAnalogInputs\(\)](#) - VI2 platform
- Bugfix for CFGIN\_NOT\_IN\_USE - VI2 platform
- Added examples for [SoftKey](#) class - VI2 platform

### 1.2.13 Version 2.14.0.23 - XM9 platform

- See separate release notes.

### 1.2.14 Version 2.14.0.0 - all platforms

- New function [About.getIsPCBBluetoothAvailable\(\)](#) for reading PCB Bluetooth chip status from EEPROM - VI2 platform
- Bugfix for [About.getNrOfButtons\(\)](#) - VI2 platform
- Bugfix for opening files from read only file system - All platforms

### 1.2.15 Version 2.13.2.25 - XM9 platform

- See separate release notes.

### 1.2.16 Version 2.13.0.0 - VS and VI2 platforms

- Initial support for VI2 platform
- New class [SoftKey](#) for VI2 platform pushbutton's status and backlight
- Added support for [AuxVersion.getFPGAVersion\(\)](#) - VS platform

### 1.2.17 Version 2.12.0.0 - VS platform

- New functions [Buzzer\\_getScaledVolume\(\)](#), [Buzzer\\_setScaledVolume\(\)](#) for all platforms. Replaces [Buzzer\\_getVolume\(\)](#), [Buzzer\\_setVolume\(\)](#) which may be removed in a future version.
- [CrossControl](#) Software License Agreement replaces the Maximatecc Software License Agreement.

### 1.2.18 Version 2.11.0.0 - VS platform

- New functions [Config\\_getShortBeepSettings\(\)](#), [Config\\_setShortBeepSettings\(\)](#), [Config\\_getLongBeepSettings\(\)](#), [Config\\_setLongBeepSettings\(\)](#), [FrontLED\\_getBootLEDConfig\(\)](#), [FrontLED\\_setBootLEDConfig\(\)](#), [FrontLED\\_getPostBootLEDConfig\(\)](#) and [FrontLED\\_setPostBootLEDConfig\(\)](#) for VS platform.
- [About\\_getNrOfUSBConnections\(\)](#) now reports correct number of USB ports for VS platform.
- [About\\_hasOsBooted\(\)](#) support added for VS platform.

### 1.2.19 Version 2.10.0.0 - VS platform

- Initial support for VS platform.
- New functions [DigIO\\_getDigPowerOutput\(\)](#), [DigIO\\_setDigPowerOutput\(\)](#), [Config\\_getDigPowerOutputStartupC](#) and [Config\\_setDigPowerOutputStartupConfig\(\)](#) for VS platform.

### 1.2.20 Version 2.9.0.0 - XA/XS/VC/VA and XM2 Linux platforms

- The entire api code base is converted to use typedefs for integer and float types (`int8_t`, `int16_t` etc.), defined in [CCAuxTypes.h](#). This may affect your build if you use functions that previously had "unsigned long" arguments. They should be converted to "unsigned int" or "uint32\_t".
- Memory leaks, security and reliability fixes.
- `ccsettingsconsole`: Limited backlight and lightsensor support.
- Improved stability of firmware updates.

### 1.2.21 Version 2.8.3.0 - VC/VA Linux platforms

- Support for VA platform.
- Code corrections based on static analysis which should lead to improved reliability.
- [Config\\_getOnOffSignalState\(\)](#) API function added.
- [Config\\_getOnOffTriggerMode\(\)](#) and [Config\\_setOnOffTriggerMode\(\)](#) API functions added.
- Using modified `stm32flash` for `stm32` flashing in Linux.

- The function [Telematics\\_getGPSAntennaStatus\(\)](#) is not supported in the XA/XS platform.
- The function [Telematics\\_getGPSAntennaStatus\(\)](#) is only supported on revision A XM AI add-on boards.

### 1.2.22 Version 2.7.4.0 - VC Linux platform

- Bugfixes and documentation updates.

### 1.2.23 Version 2.7.3.0 - VC Linux platform

- Added functions [CfgIn\\_getMinFrequencyThreshold\(\)](#) and [CfgIn\\_setMinFrequencyThreshold\(\)](#).
- Added restrictions on the usage of start-up triggers in combination with button configurations.

### 1.2.24 Version 2.7.2.0 - XM/XL Windows x86, x64 platform, VC Linux platform

- XM/XL, Windows: Fixed a bug introduced in 2.7.0.0 where the light sensor data could not be read on XM/XL.
- CCSettingsConsole: Fixed an issue where some commands did not work in Windows.

### 1.2.25 Version 2.7.1.0 - XM/XL Windows x86, x64 platform, VC Linux platform

- ccvideo: Fixed an issue where channels were not displayed correctly in the menu.

### 1.2.26 Version 2.7.0.0 - XM/XL Windows x86, x64 platform, VC Linux platform

- VC: Support for the VC platform (Linux).
- XM/XL: Support for the XM 2.0 platform (Windows/Linux).
- Added the following classes/functions for the VC platform:
  - Class [CfgIn](#) - Functions for managing configurable inputs
  - Class [PWMOut](#) - Functions for managing PWM outputs
  - [About\\_getNrOfCfgInConnections\(\)](#)
  - [About\\_getNrOfPWMOutConnections\(\)](#)
  - [About\\_getNrOfButtons\(\)](#)
  - [About\\_getNrOfButtons\(\)](#)

- [Config\\_getButtonFunction\(\)](#)
  - [Config\\_setButtonFunction\(\)](#)
- Added the following functions for all platforms:
  - [About\\_getUserEepromData\(\)](#)
  - [About\\_setUserEepromData\(\)](#)
  
- Known issues:
  - XA/XS: Same as 2.4.7.0 release
  - XM/XL: Same as 2.5.0.0 release
  - VC: -

### 1.2.27 Version 2.6.2.0 - XM/XL Windows x86, x64 platform

- XM/XL: Fix for an issue with the function [Video\\_getActiveChannel\(\)](#) in x86 API on x64 OS.
  
- XM/XL: Support for [Power\\_getCanOCDStatus\(\)](#) and [Power\\_getVideoOCDStatus\(\)](#) with SS v1.2.0.0 or later.
- XM/XL: Support for optional integrated WLAN on CCpilot XL4.
- XM/XL: CCsettings: Improved Telematic GUI when not all interfaces are available.
- XM/XL: SnbService: Improved unit type descriptions: "CCpilot XM" instead of just "XM".
- XM/XL: CCsettings, CCvideo and TouchCalibrator: QT x86 libraries updated to v4.8.5.
  
- Known issues:
  - XA/XS: Same as 2.4.7.0 release
  - XM/XL: Same as 2.5.0.0 release

### 1.2.28 Version 2.6.1.0 - XM/XL Windows x86, x64 platform and XA/XS Linux platform

- XA/XS: Functions added: [Video\\_getGraphicsOverlay\(\)](#) and [Video\\_setGraphicsOverlay\(\)](#).
- XM/XL: 64-bit support. Both x86 and x64 versions of the API can be installed at the same time on x64 systems.
- XM/XL: SnbService is now a selectable component in the installer.

- XM/XL: CCsettings: Factory default settings for XL4 updated: ShortButton↵ PressAction=ActionShutDown, OnOffSigAction=NoAction
- Known issues:
  - XA/XS: Same as 2.4.7.0 release
  - XM/XL: Same as 2.5.0.0 release

### 1.2.29 Version 2.5.0.0 - XM/XL x86 platform

- CCAux2 API: Support for the XL platform. The XL platform is almost identical to the XM platform in terms of API support.
- CCAux2 API: Added SMART support for a second card used in XL (new functions [Smart\\_getRemainingLifeTime2\(\)](#), [Smart\\_getDeviceSerial2\(\)](#) and [Smart\\_getInitialTime2\(\)](#)).
- CCAux2 API: Bugfix for crash when incorrect filename was supplied to the functions [FirmwareUpgrade.startFpgaUpgrade\(\)](#) and [FirmwareUpgrade.startFpgaVerification\(\)](#).
- CCvideo: Fixed a bug where selecting video 3 and 4 both selected video 3. The bug was only present in CCvideo v2.4.0.0 for XM and not in previous versions.
- CCvideo,CCAuxDrv: On the XL platform, video channel 3 and 4 are not available on both devices as on XM. Instead ch1 and ch2 can be selected for both devices.  
Only one channel can be shown at the same time per device and a device is on the XL platform equal to a physical connector.
- CCAux2CS: Added support for SMART interface for the C# dll
- CCAux2CS: Rewrote the following functions and changed their declaration to use System.String as output. The old overloads now return ERR\_NOT\_SUPPORTED↵ :
  - [About\\_getMainPCBSerial\(\)](#)
  - [About\\_getUnitSerial\(\)](#)
  - [About\\_getMainPCBArt\(\)](#)
  - [About\\_getMainManufacturingDate\(\)](#)
  - [About\\_getMainHWversion\(\)](#)
  - [About\\_getMainProdRev\(\)](#)
  - [About\\_getMainProdArtNr\(\)](#)
  - [About\\_getAddOnPCBSerial\(\)](#)
  - [About\\_getAddOnPCBArt\(\)](#)
  - [About\\_getAddOnManufacturingDate\(\)](#)
  - [About\\_getAddOnHWversion\(\)](#)
  - [FirmwareUpgrade.startFpgaUpgrade\(\)](#)
  - [FirmwareUpgrade.startFpgaVerification\(\)](#)
  - [FirmwareUpgrade.startSSUpgrade\(\)](#)
  - [FirmwareUpgrade.startSSVerification\(\)](#)
  - [FirmwareUpgrade.startFrontUpgrade\(\)](#)
  - [FirmwareUpgrade.startFrontVerification\(\)](#)



### [Video\\_takeSnapshot\(\)](#)

- Known issues:
  - Some API functions are missing from ccsettingsconsole and CCAux2CS.

## **1.2.30 Version 2.4.7.0 - XM Linux platform**

- XM: Improved fault-handling in function [PowerMgr\\_registerControlledSuspendOrShutDown\(\)](#)

- Known issues:
  - Same as 2.4.6.0 release

## **1.2.31 Version 2.4.6.0 - XA/XS platform**

- XA/XS: Improve initialization of video channels 3/4
- XA/XS: Prevent scrolling when changing between video channels 3/4
- Calling [Buzzer\\_buzze\(\)](#) no longer leaks memory

- Known issues:
  - Same as 2.4.0.0 release (minus Buzzer\_buzze memory leak)

## **1.2.32 Version 2.4.2.0 - XA/XS platform**

- XA/XS: [Config\\_getRS485Enabled\(\)](#) and [Config\\_setRS485Enabled\(\)](#) now uses settings file for intermediate storage

- Known issues:
  - Same as 2.4.0.0 release

## **1.2.33 Version 2.4.0.0 - XA/XS, XM platforms**

- Removed the following functions: Config\_get/set TFT Mode/Scan/Mirror
- Optimized version queries of different firmware components
- Bugfixes for [Backlight](#) and [Light sensor](#)
- The factory defaults settings in CCsettings no longer generates errors
- CCSettings and StartupGUI rebranded for maximatecc

- CCSettings now adapts to the number of CAN ports available
- Added the following function blocks: `Battery`, `PowerMgr` and `SmartSmart` from 1.x API
- XM: CCAux2 is now fully supported on the XM platform with the same functionality as in the 1.6.4.0 release.
- XM: CCAux api 1.6.4.0 will be available for backwards compatibility. It is compatible back to the 1.3.1.0 release.
- XA/XS: `Config_setRS485Enabled()` now sets MP\_RS422\_MODE GPIO pins to correct state
- XA/XS: `Video_setMirroring()` implemented
- XA/XS: Playing two video channels simultaneously now works (1/2+3/4)
- XA/XS: Video can be cropped from left/right for channels 3/4
- XA/XS: Various other improvements for video channels 3/4
- XA/XS: Video standard now reported correctly
- XA/XS: ccvideo context menu now appearing consistently
- XA/XS: ccvideo context menu hanging now fixed
- XA/XS: ccvideo blanking now fixed
- XA/XS: ccvideo now handles rotation
- XA/XS: ccsettingsconsole now up to date
- XA/XS: Context menu no longer opened while calibrating
- XA/XS: The PowerOnAtStartup setting ("Always start when power turned on" in CCsettings) was always read as Enabled
- XA/XS: 1V2 is now a supported ADC channel on some instances
- XA/XS: Added TS\_TCHAUTOCAL in TouchScreen class
- ccauxd: Fixed issues that caused crash when shutting the daemon off
- ccauxd: Added support for PowerMgr
  
- Known issues:
  - XA/XS: When automatic backlight is enabled, updating SS or Front uC software is very slow and may fail. Workaround: Make sure automatic backlight is disabled before attempting to do any firmware upgrade.
  - XA/XS: CCSettings - Advanced: After Firmware update, the shutdown button does not work. Workaround: Turn off power to the device.
  - Some info/functions are missing from ccsettingsconsole
  - XA/XS: `About_hasOsBooted()` can return true even when not all drivers have not been loaded (API)
  - XA/XS: Calling `Buzzer_buzze()` in non-blocking mode leaks memory

### 1.2.34 Version 2.3.0.0 - XA/XS platform

- Functions added: [Backlight\\_getHWStatus\(\)](#), [Config\\_getRS485Enabled\(\)](#) and [Config\\_setRS485Enabled\(\)](#)
- CCSettings: Led tab improved
- CCSettings: Hide unsupported options in Power tab
- CCSettings: Hide suspend options if unsupported by HW
- CCSettings: Fixed rotation glitches
- Bugfixes
  
- Known issues:
  - Same as 2.2.0.0 release

### 1.2.35 Version 2.2.0.0 - XA/XS platform

- Functions added: [About\\_getIsAnybusMounted\(\)](#), [Config\\_setTFTMode\(\)](#), [Config\\_getTFTMode\(\)](#), [Video\\_showFrame\(\)](#) and [About\\_getIOExpanderValue\(\)](#)
- Fixed rotation issues with GUI applications
- Many bugfixes
  
- Known issues:
  - When automatic backlight is enabled, updating SS or Front uC software is very slow and may fail. Workaround: Make sure automatic backlight is disabled before attempting to do any firmware upgrade.
  - CCSettings - Advanced: After Firmware update, the shutdown button does not work. Workaround: Turn off power to the device.
  - Some info/functions are missing from ccsettingsconsole
  - [About\\_hasOsBooted\(\)](#) can return true even when not all drivers have not been loaded (API)
  - Calling [Buzzer\\_buzze\(\)](#) in non-blocking mode leaks memory
  - ccvideo: Rightclick (long press) menu not appearing consistently
  - Calling [Video\\_showVideo\(\)](#) for ports 3/4 will not return if no camera is attached
  - Cannot show analog video from two ports simultaneously (1/2+3/4), trying to do so leads to crash
  - For ports 3/4, video sometimes scrolls or has wrong size when starting the application first time
  - API calls for analog video currently not supported: [get/setMirroring](#), [get/setCropping](#) (for ports 3/4), [get/setDeInterlaceMode](#), [get/setScaling](#), [get/setColorKeys](#)
  - ccvideo: Selecting "Mirror image" does not have an effect

### 1.2.36 Version 2.1.0.0 - XA/XS platform

- Functions added: [Power\\_getVideoOCDStatus\(\)](#), [Power\\_getCanOCDStatus\(\)](#) and [About\\_hasOsBooted\(\)](#)
- Touch calibration can be started from CCSettings
- 7" touch calibration now supported
- Many bugfixes
  
- Known issues:
  - [About\\_hasOsBooted\(\)](#) can return true even when not all drivers have not been loaded
  - Analog video API only supports VIDEO1/2 ports
  - Video control only supports positioning and resizing
  - The factory defaults button in the Advanced tab in CCSettings produces some error messages. These can be ignored

### 1.2.37 Version 2.0.0.0 - XA/XS platform

- Initial release
- The CCAux API v1.x from the CCPilot XM platform has been rewritten to ensure compability between releases
- Porting to CCPilot XA/XS platform nearly complete. Some new platform specific functions remain to be implemented
- The API gives access to several hardware interfaces, for example backlight, buzzer, diagnostics, frontled, lightsensor and analog video interfaces
  
- Known issues:
  - Digital input/output does not work correctly
  - CAN settings interface does not work
  - Analog video API only supports VIDEO1/2 ports
  - Video control only supports positioning and resizing
  - SS/Front software update - sometimes crashes before update has begun. When this happens (segmentation fault or Open failed error), restart the unit and try again
  - Font issue in CCSettings causes some text to disappear
  - TouchCalibrator cannot be started from within CCSettings. Instead it can be started manually: # TouchCalibrator -qws

- The factory defaults button in the Advanced tab in CCSettings produces some error messages. These can be ignored
- Error messages related to automatic backlight will show the very first time the Display tab in CCsettings is opened. These can be ignored.
- GetHWErrorStatusString functions do not return correct description of error messages

### 1.3 Known Issues

- XA/XS: Unsupported API calls for analog video: get/setDeInterlaceMode, get/set↔Scaling, get/setColorKeys, get/setCropping (for ports 3/4)
- XA/XS: ccvideostream: de-interlacing artifacts with certain output window sizes

## Chapter 2

# Functions added in v2.17.0.0

[AuxVersion.getCanFdApiVersion\(\)](#)  
[AuxVersion.getFullOSVersion\(\)](#)  
[AuxVersion.getFullCCAuxVersion\(\)](#)

## Chapter 3

# Functions added in v2.16.2.0

[CanSetting\\_getFlexibleDataBaudrates\(\)](#)  
[CanSetting\\_setFlexibleDataBaudrates\(\)](#)  
[AuxVersion\\_getCanFdMcuVersion\(\)](#)

## **Chapter 4**

# **Functions added in v2.15.0.0**

[About\\_getNrOfAnalogInputs\(\)](#)



## **Chapter 5**

# **Functions added in v2.14.0.0**

[About\\_getIsPCBBluetoothAvailable\(\)](#)

## Chapter 6

### Functions added in v2.13.0.0

GetSoftKey()  
SoftKey\_release()  
SoftKey\_getStatus()  
SoftKey\_setBacklightIntensity()  
SoftKey\_getBacklightIntensity()  
SoftKey\_getBacklightSignal()  
SoftKey\_getBacklightOnTime()  
SoftKey\_getBacklightOffTime()  
SoftKey\_getBacklightIdleTime()  
SoftKey\_getBacklightNrOfPulses()  
SoftKey\_setBacklightSignal()  
SoftKey\_setBacklightOnTime()  
SoftKey\_setBacklightOffTime()  
SoftKey\_setBacklightIdleTime()  
SoftKey\_setBacklightNrOfPulses()  
SoftKey\_setBacklightOff()  
SoftKey\_getBacklightEnabledDuringStartup()  
SoftKey\_setBacklightEnabledDuringStartup()  
SoftKey\_setMultipleBacklightIntensities()  
SoftKey\_getMultipleBacklightIntensities()  
SoftKey\_getBootBacklightConfig()  
SoftKey\_setBootBacklightConfig()

## Chapter 7

# Functions added in v2.12.0.0

[Buzzer\\_getScaledVolume\(\)](#)  
[Buzzer\\_setScaledVolume\(\)](#)

## Chapter 8

### Functions added in v2.11.0.0

`Config_getShortBeepSettings()`  
`Config_setShortBeepSettings()`  
`Config_getLongBeepSettings()`  
`Config_setLongBeepSettings()`  
`FrontLED_getBootLEDConfig()`  
`FrontLED_setBootLEDConfig()`  
`FrontLED_getPostBootLEDConfig()`  
`FrontLED_setPostBootLEDConfig()`

## Chapter 9

### Functions added in v2.10.0.0

DigIO\_getDigPowerOutput()  
DigIO\_setDigPowerOutput()  
Config\_getDigPowerOutputStartupConfig()  
Config\_setDigPowerOutputStartupConfig()

## Chapter 10

### Functions added in v2.8.3.0

Config\_getOnOffSignalState()  
CfgIn\_getFrequencyValue()  
CfgIn\_setFrequencyFilterLevel()  
TouchScreenCalib\_autoSensorCalib()  
Config\_setOnOffTriggerMode()  
Config\_getOnOffTriggerMode()

# Chapter 11

## Functions added in v2.7.3.0

[CfgIn\\_getMinFrequencyThreshold\(\)](#)  
[CfgIn\\_setMinFrequencyThreshold\(\)](#)

## Chapter 12

### Functions added in v2.7.0.0

About\_getUserEepromData()  
About\_setUserEepromData()  
About\_getNrOfCfgInConnections()  
About\_getNrOfPWMOutConnections()  
About\_getNrOfButtons()  
Config\_getButtonFunction()  
Config\_setButtonFunction()  
GetCfgIn()  
CfgIn\_release()  
CfgIn\_setCfgInMode()  
CfgIn\_getCfgInMode()  
CfgIn\_getValue()  
CfgIn\_getPwmValue()  
GetPWMOut()  
PWMOut\_release()  
PWMOut\_setPWMOutputChannelDutyCycle()  
PWMOut\_setPWMOutputChannelFrequency()  
PWMOut\_getPWMOutputChannelDutyCycle()  
PWMOut\_getPWMOutputChannelFrequency()  
PWMOut\_getPWMOutputStatus()  
PWMOut\_setPWMOutOff()



## Chapter 13

# Functions added in v2.6.1.0

[Video\\_getGraphicsOverlay\(\)](#)  
[Video\\_setGraphicsOverlay\(\)](#)

## Chapter 14

### Functions added in v2.5.0.0

[Smart\\_getRemainingLifeTime2\(\)](#)  
[Smart\\_getDeviceSerial2\(\)](#)  
[Smart\\_getInitialTime2\(\)](#)

# Chapter 15

## Module Index

### 15.1 Modules

Here is a list of all modules:

About functions . . . . .	31
Adc functions . . . . .	52
AuxVersion functions . . . . .	56
Backlight functions . . . . .	65
Battery functions . . . . .	73
Buzzer functions . . . . .	83
CanSetting functions . . . . .	89
CfgIn functions . . . . .	93
Config functions . . . . .	101
Diagnostic functions . . . . .	125
DigIO functions . . . . .	131
FirmwareUpgrade functions . . . . .	135
FrontLED functions . . . . .	145
Lightsensor functions . . . . .	157
Power functions . . . . .	162
PowerMgr functions . . . . .	170
PWMOut functions . . . . .	179
Smart functions . . . . .	184
SoftKey functions . . . . .	191
Telematics functions . . . . .	205
TouchScreen functions . . . . .	216
TouchScreenCalib functions . . . . .	222
Video functions . . . . .	226

# Chapter 16

## Namespace Index

### 16.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">CrossControl</a> .....	245
------------------------------------	-----

# Chapter 17

## Data Structure Index

### 17.1 Data Structures

Here are the data structures with brief descriptions:

BatteryTimerType	291
BuzzerSetup	292
FpgaLedTimingType	293
LedColorMixType	294
LedTimingType	294
received_video	295
TimerType	296
UpgradeStatus	297
version_info	297
video_dec_command	298

# Chapter 18

## File Index

### 18.1 File List

Here is a list of all files with brief descriptions:

IncludeFiles/About.h	300
IncludeFiles/Adc.h	302
IncludeFiles/AuxVersion.h	302
IncludeFiles/Backlight.h	303
IncludeFiles/Battery.h	304
IncludeFiles/Buzzer.h	305
IncludeFiles/CanSetting.h	306
IncludeFiles/CCAuxErrors.h	307
IncludeFiles/CCAuxTypes.h	307
IncludeFiles/CCPlatform.h	311
IncludeFiles/CfgIn.h	311
IncludeFiles/Config.h	312
IncludeFiles/Diagnostic.h	314
IncludeFiles/DiagnosticCodes.h	315
IncludeFiles/DigIO.h	316
IncludeFiles/FirmwareUpgrade.h	316
IncludeFiles/FrontLED.h	317
IncludeFiles/Lightsensor.h	319
IncludeFiles/Power.h	319
IncludeFiles/PowerMgr.h	320
IncludeFiles/PWMOut.h	321
IncludeFiles/Smart.h	321
IncludeFiles/SoftKey.h	322
IncludeFiles/Telematics.h	323
IncludeFiles/TouchScreen.h	324
IncludeFiles/TouchScreenCalib.h	325
IncludeFiles/Video.h	326

# Chapter 19

## Module Documentation

### 19.1 About functions

#### Functions

- EXTERN\_C CCAUXDLL\_API [ABOUTHANDLE](#) CCAUXDLL\_CALLING\_CONV [GetAbout](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [About\\_release](#) ([ABOUTHANDLE](#))
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getMainPCBSerial](#) ([ABOUTHANDLE](#), [char.t](#) \*buff, [int32.t](#) len)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getUnitSerial](#) ([ABOUTHANDLE](#), [char.t](#) \*buff, [int32.t](#) len)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getMainPCBArt](#) ([ABOUTHANDLE](#), [char.t](#) \*buff, [int32.t](#) length)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getMainManufacturingDate](#) ([ABOUTHANDLE](#), [char.t](#) \*buff, [int32.t](#) len)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getMainHWversion](#) ([ABOUTHANDLE](#), [char.t](#) \*buff, [int32.t](#) len)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getMainProdRev](#) ([ABOUTHANDLE](#), [char.t](#) \*buff, [int32.t](#) len)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getMainProdArtNr](#) ([ABOUTHANDLE](#), [char.t](#) \*buff, [int32.t](#) len)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getNrOfETHConnections](#) ([ABOUTHANDLE](#), [uint8.t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getNrOfCANConnections](#) ([ABOUTHANDLE](#), [uint8.t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getNrOfVideoConnections](#) ([ABOUTHANDLE](#), [uint8.t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getNrOfUSBConnections](#) ([ABOUTHANDLE](#), [uint8.t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getNrOfSerialConnections](#) ([ABOUTHANDLE](#), [uint8.t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [About\\_getNrOfDigIOConnections](#) ([ABOUTHANDLE](#), [uint8.t](#) \*NrOfConnections)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsDisplayAvailable](#) (ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsTouchScreenAvailable](#) (ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsPCBBluetoothAvailable](#) (ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getDisplayResolution](#) (ABOUTHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnPCBSerial](#) (ABOUTHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnPCBArt](#) (ABOUTHANDLE, char\_t \*buff, int32\_t length)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnManufacturingDate](#) (ABOUTHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnHWversion](#) (ABOUTHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsWLANMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsGPSMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsGPRSMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsBTMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getFrontPcbRev](#) (ABOUTHANDLE, uint8\_t \*major, uint8\_t \*minor)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsIOExpanderMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIOExpanderValue](#) (ABOUTHANDLE, uint16\_t \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_hasOsBooted](#) (ABOUTHANDLE, bool \*bootComplete)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsAnybusMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfCfgInConnections](#) (ABOUTHANDLE, uint8\_t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfPWMOutConnections](#) (ABOUTHANDLE, uint8\_t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfButtons](#) (ABOUTHANDLE, int32\_t \*numbuttons)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfAnalogInputs](#) (ABOUTHANDLE, int32\_t \*numanalogins)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getUserEepromData](#) (ABOUTHANDLE, char\_t \*buff, uint16\_t length)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_setUserEepromData](#) (ABOUTHANDLE, uint16\_t startpos, const char\_t \*buff, uint16\_t length)



### 19.1.1 Detailed Description

Functions in the About class

### 19.1.2 Function Documentation

#### About\_getAddOnHWversion()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::About_getAddOnHWversion (
    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get Add on hardware version.

Supported Platform(s): XL, XM, XS, XA

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. See the enum eErr for details.

Example Usage:

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

#### About\_getAddOnManufacturingDate()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::About_getAddOnManufacturingDate (
    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get Add on manufacturing date.

Supported Platform(s): XL, XM, XS, XA

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. See the enum eErr for details.

**Example Usage:**

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

**About\_getAddOnPCBArt()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::About_getAddOnPCBArt (
```

```
    ABOUTHANDLE ,
    char_t * buff,
    int32_t length )
```

Get Add on PCB article number.

Supported Platform(s): XL, XM, XS, XA

**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. See the enum eErr for details.

**Example Usage:**

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

**About\_getAddOnPCBSerial()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::About_getAddOnPCBSerial (
```

```
    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get Add on PCB serial number.

Supported Platform(s): XL, XM, XS, XA

**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. See the enum eErr for details.

**Example Usage:**

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

**About\_getDisplayResolution()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::About_getDisplayResolution (
    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get display resolution.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**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 display resolution will be returned in the format "1024x768"

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = About_getDisplayResolution (pAbout, buffer, buffer_len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Display resolution: " << buffer << endl;
```

**About\_getFrontPcbRev()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::About_getFrontPcbRev (
    ABOUTHANDLE ,
    uint8_t * major,
    uint8_t * minor )
```

Get the front hardware pcb revision in the format major.minor (e.g. 1.1).

Supported Platform(s): XA, XS

**Parameters**

<i>major</i>	The major pcb revision.
<i>minor</i>	The minor pcb revision.

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**About\_getIOExpanderValue()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIOExpander↵
Value (
```

```
    ABOUTHANDLE ,
    uint16_t * value )
```

Get Value for IO Expander

Supported Platform(s): XA, XS

**Parameters**

<i>value</i>	IO Expander value.
--------------	--------------------

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**About\_getIsAnybusMounted()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIs↵
AnybusMounted (
```

```
    ABOUTHANDLE ,
    bool * mounted )
```

Get Anybus mounting status.

Supported Platform(s): XA, XS

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
bool isAnybusMounted;
err = CrossControl::About_getIsAnybusMounted(pAbout, &isAnybusMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Anybus mounted: " << (isAnybusMounted ? "YES" : "NO") << endl;
```

**About\_getIsBTMounted()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIsBTMounted (
    ABOUTHANDLE ,
    bool * mounted )
```

Get BlueTooth module mounting status.

Supported Platform(s): XL, XM, XS, XA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

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

**About\_getIsDisplayAvailable()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIsDisplayAvailable (
    ABOUTHANDLE ,
    bool * available )
```

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

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

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

**About\_getIsGPRSMounted()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIsGPRSMounted (
```

```

    ABOUTHANDLE ,
    bool * mounted )

```

Get GPRS module mounting status.

Supported Platform(s): XL, XM, XS, XA

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```

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

```

### About\_getIsGPSMounted()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIsGPSMounted (

```

```

    ABOUTHANDLE ,
    bool * mounted )

```

Get GPS module mounting status.

Supported Platform(s): XL, XM, XS, XA

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```

bool isGPSMounted;
err = About_getIsGPSMounted (pAbout, &isGPSMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "GPS mounted: " << (isGPSMounted ? "YES" : "NO") << endl;

```

### About\_getIsIOExpanderMounted()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIsIOExpanderMounted (

```

```

    ABOUTHANDLE ,
    bool * mounted )

```

Get IO Expander mounting status.

Supported Platform(s): XA, XS

## Parameters

<i>mounted</i>	Is IO Expander mounted?
----------------	-------------------------

## Returns

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

## Example Usage:

```
bool isIOExpanderMounted;
err = CrossControl::About_getIsIOExpanderMounted(pAbout, &isIOExpanderMounted);
if (CrossControl::ERR_SUCCESS == err)
    cout << "IOExpander mounted: " << (isIOExpanderMounted ? "YES" : "NO") <<
        endl;
```

**About\_getIsPCBBluetoothAvailable()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIsPCBBluetoothAvailable (
    ABOUTHANDLE ,
    bool * available )
```

Get PCB mounted bluetooth status. Not to be confused with the bluetooth module mounted on the telematics add on board

Supported Platform(s): VI2

## Parameters

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

## Returns

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

## Example Usage:

```
bool pcbBluetoothAvailable;
err = About_getIsPCBBluetoothAvailable (pAbout, &pcbBluetoothAvailable);
if (CrossControl::ERR_SUCCESS == err)
    cout << "PCB Bluetooth available: " << (pcbBluetoothAvailable ? "YES" : "NO")
        << endl;
```

**About\_getIsTouchScreenAvailable()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIsTouchScreenAvailable (
    ABOUTHANDLE ,
    bool * available )
```

Get Display TouchScreen status.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700



## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

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

**About\_getIsWLANMounted()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIsWLANMounted (
    ABOUTHANDLE ,
    bool * mounted )
```

Get WLAN module mounting status.

Supported Platform(s): XL, XM, XS, XA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

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

**About\_getMainHWversion()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getMainHWversion (
    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get main hardware version (PCB revision).

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## 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. See the enum eErr for details.

## Example Usage:

```
err = About_getMainHWversion (pAbout, buffer, buffer.len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main hardware version: " << buffer << endl;
```

**About\_getMainManufacturingDate()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getMainManufacturingDate (
    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get main manufacturing date.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## 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. See the enum eErr for details.

## Example Usage:

```
err = About_getMainManufacturingDate (pAbout, buffer, buffer.len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Manufacturing date: " << buffer << endl;
```

**About\_getMainPCBart()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getMainPCBart (
    ABOUTHANDLE ,
    char_t * buff,
    int32_t length )
```

Get main PCB article number.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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. See the enum eErr for details.

Example Usage:

```
err = About_getMainPCBArt (pAbout, buffer, buffer.len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main PCB article number: " << buffer << endl;
```

### About\_getMainPCBSerial()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::About_getMainPCBSerial (
    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get main PCB serial number.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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. See the enum eErr for details.

Example Usage:

```
err = About_getMainPCBSerial (pAbout, buffer, buffer.len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main PCB serial: " << buffer << endl;
```

### About\_getMainProdArtNr()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::About_getMainProdArtNr (
```

```

    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )

```

Get main product article number.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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. See the enum eErr for details.

#### Example Usage:

```

err = About_getMainProdArtNr (pAbout, buffer, buffer.len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main product article number: " << buffer << endl;

```

### About\_getMainProdRev()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getMainProdRev (

```

```

    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )

```

Get main product revision.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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. See the enum eErr for details.

#### Example Usage:

```

err = About_getMainProdRev (pAbout, buffer, buffer.len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Main product revision: " << buffer << endl;

```

**About\_getNrOfAnalogInputs()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfAnalogInputs (
    ABOUTHANDLE ,
    int32_t * numanalogins )
```

Get number of external analog inputs.

Supported Platform(s): VI2

Parameters

<i>numanalogins</i>	Number of external analog inputs.
---------------------	-----------------------------------

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

**About\_getNrOfButtons()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfButtons (
    ABOUTHANDLE ,
    int32_t * numbuttons )
```

Get number of configurable buttons.

Supported Platform(s): VC, VA, VS, VI2

Parameters

<i>numbuttons</i>	Number of configurable buttons.
-------------------	---------------------------------

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
int nrOfButtons;
err = About_getNrOfButtons (pAbout, &nrOfButtons);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of configurable buttons: " << (int)nrOfButtons << endl;
else if (CrossControl::ERR_NOT_SUPPORTED == err)
    cout << "About_getNrOfButtons: Not supported" << endl;
```

**About\_getNrOfCANConnections()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfCANConnections (
```

```

    ABOUTHANDLE ,
    uint8_t * NrOfConnections )

```

Get number of CAN connections present.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

<i>NrOfConnections</i>	Returns the number of connections.
------------------------	------------------------------------

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```

unsigned char nrOfCANConnections;
err = About_getNrOfCANConnections (pAbout, &nrOfCANConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of CAN connections: " << (int)nrOfCANConnections << endl;

```

### About\_getNrOfCfgInConnections()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfCfgInConnections (

```

```

    ABOUTHANDLE ,
    uint8_t * NrOfConnections )

```

Get number of configurable input connections present.

Supported Platform(s): VC, VA, VS, VI2

Parameters

<i>NrOfConnections</i>	Returns the number of inputs.
------------------------	-------------------------------

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```

unsigned char nrOfCfgIn;
err = About_getNrOfCfgInConnections (pAbout, &nrOfCfgIn);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of configurable inputs: " << (int)nrOfCfgIn << endl;
else if (CrossControl::ERR_NOT_SUPPORTED == err)
    cout << "About_getNrOfCfgInConnections: Not supported" << endl;

```

### About\_getNrOfDigIOConnections()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfDigIOConnections (

```

```

    ABOUTHANDLE ,
    uint8_t * NrOfConnections )

```

Get number of digital I/O connections present.  
Supported Platform(s): XL, XM, XS, XA

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

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

### About\_getNrOfETHConnections()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfETHConnections (
    ABOUTHANDLE ,
    uint8_t * NrOfConnections )
```

Get number of Ethernet connections present.  
Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

<i>NrOfConnections</i>	Returns the number of connections.
------------------------	------------------------------------

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
unsigned char nrOfEthConnections;
err = About_getNrOfETHConnections (pAbout, &nrOfEthConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of ethernet connections: " << (int)nrOfEthConnections << endl;
```

### About\_getNrOfPWMOutConnections()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfPWMOutConnections (
    ABOUTHANDLE ,
    uint8_t * NrOfConnections )
```

Get number of PWM Output connections present.  
Supported Platform(s): VC, VA, VI2

## Parameters

<i>NrOfConnections</i>	Returns the number of outputs.
------------------------	--------------------------------

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
unsigned char nrOfPwmOut;
err = About_getNrOfPWMOutConnections (pAbout, &nrOfPwmOut);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of PWM outputs: " << (int)nrOfPwmOut << endl;
else if (CrossControl::ERR_NOT_SUPPORTED == err)
    cout << "About_getNrOfPWMOutConnections: Not supported" << endl;
```

**About\_getNrOfSerialConnections()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfSerialConnections (
    ABOUTHANDLE ,
    uint8_t * NrOfConnections )
```

Get number of serial port (RS232) connections present.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

<i>NrOfConnections</i>	Returns the number of connections.
------------------------	------------------------------------

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
unsigned char nrOfSerialConnections;
err = About_getNrOfSerialConnections (pAbout, &nrOfSerialConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of serial connections: " << (int)nrOfSerialConnections << endl;
```

**About\_getNrOfUSBConnections()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfUSBConnections (
    ABOUTHANDLE ,
    uint8_t * NrOfConnections )
```

Get number of USB connections present.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700



## Parameters

<i>NrOfConnections</i>	Returns the number of connections.
------------------------	------------------------------------

## Returns

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

## Example Usage:

```
unsigned char nrOfUSBConnections;
err = About_getNrOfUSBConnections (pAbout, &nrOfUSBConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of USB connections: " << (int)nrOfUSBConnections << endl;
```

**About\_getNrOfVideoConnections()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNrOfVideoConnections (
    ABOUTHANDLE ,
    uint8_t * NrOfConnections )
```

Get number of Video connections present.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

<i>NrOfConnections</i>	Returns the number of connections.
------------------------	------------------------------------

## Returns

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

## Example Usage:

```
unsigned char nrOfVideoConnections;
err = About_getNrOfVideoConnections (pAbout, &nrOfVideoConnections);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Nr of video connections: " << (int)nrOfVideoConnections << endl;
```

**About\_getUnitSerial()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getUnitSerial (
    ABOUTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get unit serial number.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## 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. See the enum eErr for details.

## Example Usage:

```
err = About_getUnitSerial (pAbout, buffer, buffer.len);
if (CrossControl::ERR_SUCCESS == err)
    cout << "Unit serial: " << buffer << endl;
```

**About\_getUserEepromData()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getUser↔
EepromData (
    ABOUTHANDLE ,
    char_t * buff,
    uint16_t length )
```

Get User Eeprom data. The user eeprom holds 4096 bytes of data which are fully accessible. Data is always read from position 0.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA

## Parameters

<i>buff</i>	data buffer.
<i>length</i>	data buffer length or number of data bytes to read.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code.

**About\_hasOsBooted()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_hasOs↔
Booted (
    ABOUTHANDLE ,
    bool * bootComplete )
```

Get the status of the OS boot process. In Linux, drivers may be delay-loaded at start-up. If the application is started early in the boot-process, this function can be used to determine when full functionality can be obtained from the API/drivers.

Supported Platform(s): XA, XS, VC, VA, VI2, VS, V700

## Parameters

<i>bootComplete</i>	Is the OS fully booted?
---------------------	-------------------------

## Returns

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

## Example Usage:

```
bool isBootComplete;
err = CrossControl::About_hasOsBooted(pAbout, &isBootComplete);
if (CrossControl::ERR_SUCCESS == err)
    cout << "System bootup complete: " << (isBootComplete ? "YES" : "NO") << endl;
```

**About\_release()**

```
EXTERN_C CCAUXDLLAPI void CCAUXDLLCALLING_CONV CrossControl::About_release
(
    ABOUTHANDLE )
```

Delete the About object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Returns

-

## Example Usage:

```
ABOUTHANDLE pAbout = ::GetAbout();
assert(pAbout);
list.about.information(pAbout);
About_release(pAbout);
```

**About\_setUserEepromData()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::About_setUserEepromData (
```

```
    ABOUTHANDLE ,
    uint16_t startpos,
    const char_t * buff,
    uint16_t length )
```

Set User Eeprom data. The user eeprom holds 4096 bytes of data which are fully accessible.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA

## Parameters

<i>startpos</i>	eeprom write start position.
<i>buff</i>	data buffer.
<i>length</i>	buffer length to write.

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code.

**GetAbout()**

```
EXTERN_C CCAUXDLL_API ABOUTHANDLE CCAUXDLL_CALLING_CONV CrossControl::GetAbout (
    void )
```

Factory function that creates instances of the About object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Returns**

ABOUTHANDLE to an allocated About object. The returned handle needs to be deallocated using the [About.release\(ABOUTHANDLE\)](#) 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);
About_release (pAbout);
```

## 19.2 Adc functions

### Functions

- EXTERN\_C CCAUXDLL\_API ADCHANDLE CCAUXDLL\_CALLING\_CONV [GetAdc](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Adc\\_release](#) (ADCHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Adc\\_getVoltage](#) (ADCHANDLE, VoltageEnum selection, float64\_t \*value)

### 19.2.1 Detailed Description

Functions in the Adc class

### 19.2.2 Function Documentation

**Adc\_getVoltage()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Adc_getVoltage (
    ADCHANDLE ,
    VoltageEnum selection,
    float64_t * value )
```

Read measured voltage.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

<i>selection</i>	The type of voltage to get.
<i>value</i>	Voltage value in Volt. Can be undefined if return value is error code. Not all values are supported on all platforms, ERR_NOT_SUPPORTED will indicate that.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
err = Adc.getVoltage(pAdc, selection, &voltage);
if (err == CrossControl::ERR_SUCCESS)
{
    cout << left << setw(7) << description << ": " <<
        fixed << setprecision(2) << voltage << "V" << endl;
}
else if (err == CrossControl::ERR_NOT_SUPPORTED)
{
    /* Don't print anything */
}
else
{
    cout << left << setw(7) << description << ": " <<
        fixed << setprecision(2) << CrossControl::GetErrorStringA(err) << endl;
}
```

**Adc.release()**

```
EXTERN_C CCAUXDLLAPI void CCAUXDLLCALLING_CONV CrossControl::Adc.release
(
    ADCHANDLE )
```

Delete the ADC object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## 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);
output_voltage(pAdc, "VREF_INT", CrossControl::VOLTAGE_VREF_INT);
output_voltage(pAdc, "24V_BACKUP", CrossControl::VOLTAGE_24V_BACKUP);
```

```

output.voltage (pAdc, "2V5",      CrossControl::VOLTAGE_2V5);
output.voltage (pAdc, "1V1",      CrossControl::VOLTAGE_1V1);
output.voltage (pAdc, "1V3_PER",  CrossControl::VOLTAGE_1V3_PER);
output.voltage (pAdc, "1V3_VDDA", CrossControl::VOLTAGE_1V3_VDDA);
output.voltage (pAdc, "3V3_STBY", CrossControl::VOLTAGE_3V3_STBY);
output.voltage (pAdc, "VPMIC",    CrossControl::VOLTAGE_VPMIC);
output.voltage (pAdc, "VMMAIN",   CrossControl::VOLTAGE_VMMAIN);
output.voltage (pAdc, "AI.1",     CrossControl::VOLTAGE_AI.1);
output.voltage (pAdc, "AI.2",     CrossControl::VOLTAGE_AI.2);
output.voltage (pAdc, "AI.3",     CrossControl::VOLTAGE_AI.3);
Adc_release(pAdc);

```

### GetAdc()

```

EXTERN_C CCAUXDLLAPI ADCHANDLE CCAUXDLLCALLING_CONV CrossControl::GetAdc
(
    void )

```

Factory function that creates instances of the Adc object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

### Returns

ADCHANDLE to an allocated Adc object. The returned handle needs to be deallocated using the [Adc\\_release\(ADCHANDLE\)](#) 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);
output.voltage (pAdc, "VREF_INT", CrossControl::VOLTAGE_VREF_INT);
output.voltage (pAdc, "24V_BACKUP", CrossControl::VOLTAGE_24V_BACKUP);
output.voltage (pAdc, "2V5",     CrossControl::VOLTAGE_2V5);
output.voltage (pAdc, "1V1",     CrossControl::VOLTAGE_1V1);
output.voltage (pAdc, "1V3_PER", CrossControl::VOLTAGE_1V3_PER);
output.voltage (pAdc, "1V3_VDDA", CrossControl::VOLTAGE_1V3_VDDA);
output.voltage (pAdc, "3V3_STBY", CrossControl::VOLTAGE_3V3_STBY);
output.voltage (pAdc, "VPMIC",   CrossControl::VOLTAGE_VPMIC);
output.voltage (pAdc, "VMMAIN",  CrossControl::VOLTAGE_VMMAIN);
output.voltage (pAdc, "AI.1",    CrossControl::VOLTAGE_AI.1);
output.voltage (pAdc, "AI.2",    CrossControl::VOLTAGE_AI.2);
output.voltage (pAdc, "AI.3",    CrossControl::VOLTAGE_AI.3);
Adc_release(pAdc);

```

## 19.3 AuxVersion functions

### Functions

- EXTERN\_C CCAUXDLL\_API AUXVERSIONHANDLE CCAUXDLL\_CALLING\_CONV GetAuxVersion (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV AuxVersion\_release (AUXVERSIONHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getFPGAVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getSSVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getFrontVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getCCAuxVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getOSVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getCCAuxDrvVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getCanFdMcuVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getCanFdApiVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getFullIOSVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build, char\_t \*tag, uint32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getFullCCAuxVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build, char\_t \*tag, uint32\_t len)

### 19.3.1 Detailed Description

Functions in the AuxVersion class

### 19.3.2 Function Documentation



**AuxVersion\_getCanFdApiVersion()**

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::AuxVersion↔
_getCanFdApiVersion (
    AUXVERSIONHANDLE ,
    uint8_t * major,
    uint8_t * minor,
    uint8_t * release,
    uint8_t * build )

```

Get the CAN FD Windows API version.

Supported Platform(s): XL5, XM9

## 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. See the enum eErr for details.

## Example Usage:

**AuxVersion\_getCanFdMcuVersion()**

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::AuxVersion↔
_getCanFdMcuVersion (
    AUXVERSIONHANDLE ,
    uint8_t * major,
    uint8_t * minor,
    uint8_t * release,
    uint8_t * build )

```

Get the CAN FD microcontroller firmware version.

Supported Platform(s): XL5, XM9

## 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. See the enum eErr for details.

**Example Usage:**

```
err = AuxVersion.getCanFdMcuVersion(
    pAuxVersion,
    &major,
    &minor,
    &release,
    &build);
cout << setw(column_width) << "CAN FD MCU Version: ";
if (CrossControl::ERR_SUCCESS == err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;
```

**AuxVersion.getCCAuxDrvVersion()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::AuxVersion↔
_getCCAuxDrvVersion (
    AUXVERSIONHANDLE ,
    uint8_t * major,
    uint8_t * minor,
    uint8_t * release,
    uint8_t * build )
```

Get the [CrossControl](#) CCAux CCAuxDrv version. Can be used to check that the correct driver is loaded.

Supported Platform(s): XL, XM

**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. See the enum eErr for details.

**Example Usage:**

```
err = AuxVersion.getCCAuxDrvVersion(
    pAuxVersion,
    &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;

```

### AuxVersion\_getCCAuxVersion()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::AuxVersion←
_getCCAuxVersion (
    AUXVERSIONHANDLE ,
    uint8_t * major,
    uint8_t * minor,
    uint8_t * release,
    uint8_t * build )

```

Get the [CrossControl](#) CCAux API version. CCAux includes: CCAuxService/ccauxd - Windows Service/Linux daemon. CCAux2.dll/libccaux2 - The implementation of this API.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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. See the enum eErr for details.

#### Example Usage:

```

err = AuxVersion_getCCAuxVersion(
  pAuxVersion,
  &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;

```

### AuxVersion\_getFPGAVersion()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::AuxVersion←
_getFPGAVersion (
    AUXVERSIONHANDLE ,
    uint8_t * major,

```

```

uint8_t * minor,
uint8_t * release,
uint8_t * build )

```

Get the FPGA software version

Supported Platform(s): XL, XM, XS, XA, VS

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. See the enum eErr for details.

Example Usage:

```

err = AuxVersion.getFPGAVersion(
    pAuxVersion,
    &major,
    &minor,
    &release,
    &build);
cout << setw(columnWidth) << "FPGA Version: ";
if (CrossControl::ERR_SUCCESS == err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;

```

### AuxVersion\_getFrontVersion()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::AuxVersion←
_getFrontVersion (

```

```

    AUXVERSIONHANDLE ,
    uint8_t * major,
    uint8_t * minor,
    uint8_t * release,
    uint8_t * build )

```

Get the front microcontroller software version

Supported Platform(s): XL, XM, XS, XA

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. See the enum eErr for details.

**Example Usage:**

```
err = AuxVersion.getFrontVersion(
    pAuxVersion,
    &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;
```

**AuxVersion.getFullCCAuxVersion()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL-CALLING-CONV CrossControl::AuxVersion←
_getFullCCAuxVersion (
    AUXVERSIONHANDLE ,
    uint8_t * major,
    uint8_t * minor,
    uint8_t * release,
    uint8_t * build,
    char_t * tag,
    uint32_t len )
```

Get the [CrossControl](#) CCAux API version including tag name. CCAux includes← : CCAuxService/ccauxd - Windows Service/Linux daemon. CCAux2.dll/libccaux2 - The implementation of this API.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**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
<i>tag</i>	Tag name
<i>len</i>	Number of chars to read from the tag name (max 100)

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = AuxVersion.getCCAuxVersion(
    pAuxVersion,
    &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;

```

### AuxVersion\_getFullOSVersion()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::AuxVersion↵
_getFullOSVersion (
    AUXVERSIONHANDLE ,
    uint8_t * major,
    uint8_t * minor,
    uint8_t * release,
    uint8_t * build,
    char_t * tag,
    uint32_t len )

```

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

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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
<i>tag</i>	Tag name
<i>len</i>	Number of chars to read from the tag name (max 100)

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

#### Example Usage:

### AuxVersion\_getOSVersion()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::AuxVersion↵
_getOSVersion (
    AUXVERSIONHANDLE ,
    uint8_t * major,
    uint8_t * minor,

```

```
uint8_t * release,
uint8_t * build )
```

Get the **CrossControl** Operating System version.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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. See the enum eErr for details.

Example Usage:

```
err = AuxVersion.getOSVersion(
    pAuxVersion,
    &major,
    &minor,
    &release,
    &build);
cout << setw(columnWidth) << "Operating System Version: ";
if (CrossControl::ERR_SUCCESS == err)
    cout << (int) major << "." <<
        (int) minor << "." <<
        (int) release << "." <<
        (int) build << endl;
else
    cout << "unknown" << endl;
```

**AuxVersion\_getSSVersion()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::AuxVersion↵
_getSSVersion (
```

```
AUXVERSIONHANDLE ,
uint8_t * major,
uint8_t * minor,
uint8_t * release,
uint8_t * build )
```

Get the System Supervisor software version

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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. See the enum eErr for details.

**Example Usage:**

```
err = AuxVersion.getSSVersion(
    pAuxVersion,
    &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;
```

**AuxVersion.release()**

```
EXTERN_C CCAUXDLLAPI void CCAUXDLLCALLING_CONV CrossControl::AuxVersion↔
.release (
    AUXVERSIONHANDLE )
```

Delete the AuxVersion object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Returns**

-

**Example Usage:**

```
AUXVERSIONHANDLE pAuxVersion = ::GetAuxVersion();
assert (pAuxVersion);
output_versions(pAuxVersion);
AuxVersion.release(pAuxVersion);
```

**GetAuxVersion()**

```
EXTERN_C CCAUXDLLAPI AUXVERSIONHANDLE CCAUXDLLCALLING_CONV CrossControl↔
::GetAuxVersion (
    void )
```

Factory function that creates instances of the AuxVersion object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Returns**

AUXVERSIONHANDLE to an allocated AuxVersion object. The returned handle needs to be deallocated using the [AuxVersion.release\(AUXVERSIONHANDLE\)](#) 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);
AuxVersion.release(pAuxVersion);
```



## 19.4 Backlight functions

### Functions

- EXTERN\_C CCAUXDLL\_API BACKLIGHTHANDLE CCAUXDLL\_CALLING←\_CONV GetBacklight (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Backlight\_release (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getIntensity (BACKLIGHTHANDLE, uint8\_t \*intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_setIntensity (BACKLIGHTHANDLE, uint8\_t intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getStatus (BACKLIGHTHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getHWStatus (BACKLIGHTHANDLE, bool \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_startAutomaticBL (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_stopAutomaticBL (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getAutomaticBLStatus (BACKLIGHTHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_setAutomaticBLParams (BACKLIGHTHANDLE, bool bSoftTransitions)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getAutomaticBLParams (BACKLIGHTHANDLE, bool \*bSoftTransitions, float64\_t \*k)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_setAutomaticBLFilter (BACKLIGHTHANDLE, uint32\_t averageWndSize, uint32\_t rejectWndSize, uint32\_t rejectDeltaInLux, LightSensorSamplingMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getAutomaticBLFilter (BACKLIGHTHANDLE, uint32\_t \*averageWndSize, uint32\_t \*rejectWndSize, uint32\_t \*rejectDeltaInLux, LightSensorSamplingMode \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getLedDimming (BACKLIGHTHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_setLedDimming (BACKLIGHTHANDLE, CCStatus status)

### 19.4.1 Detailed Description

Functions in the Backlight class

### 19.4.2 Function Documentation

**Backlight\_getAutomaticBLFilter()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↔
_getAutomaticBLFilter (
    BACKLIGHTHANDLE ,
    uint32_t * averageWndSize,
    uint32_t * rejectWndSize,
    uint32_t * rejectDeltaInLux,
    LightSensorSamplingMode * mode )
```

Get light sensor filter parameters for automatic backlight control.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

## 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. See the enum eErr for details.

**Backlight\_getAutomaticBLParams()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↔
_getAutomaticBLParams (
    BACKLIGHTHANDLE ,
    bool * bSoftTransitions,
    float64_t * k )
```

Get parameters for automatic backlight control.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

## Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Backlight\_getAutomaticBLStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↔
_getAutomaticBLStatus (
    BACKLIGHTHANDLE ,
    uint8_t * status )
```

Get status from automatic backlight control.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Backlight\_getHWStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↔
_getHWStatus (
    BACKLIGHTHANDLE ,
    bool * status )
```

Get backlight hardware status.

**Parameters**

<i>status</i>	Backlight controller status. true: All backlight drivers works ok, false: one or more backlight drivers are faulty.
---------------	---

Supported Platform(s): XL, XM, XS, XA

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
bool backlightStatus = false;
err = Backlight_getHWStatus(pBacklight, &backlightStatus);
if (err == ERR_SUCCESS)
{
    if (backlightStatus)
        printf("Backlight hardware status: OK\n");
}
```

```

    else
        printf("Backlight hardware status: not OK, one or more backlight drivers are
            faulty\n");
    }
    else if (err == ERR_NOT_SUPPORTED)
    {
        printf("Backlight_getHWStatus: Not supported!\n");
    }
    else
    {
        printf("Error(%d) in function Backlight_getHWStatus: %s\n", err,
            GetErrorStringA(err));
    }
}

```

### Backlight\_getIntensity()

EXTERN\_C CCAUXDLLAPI eErr CCAUXDLL\_CALLING\_CONV CrossControl::Backlight↔  
\_getIntensity (

```

    BACKLIGHTHANDLE ,
    uint8_t * intensity )

```

Get backlight intensity. Note that there might be hardware limitations, limiting the minimum and/or maximum value to other than (1..255).

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### Parameters

<i>intensity</i>	The current backlight intensity (1..255).
------------------	---

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

#### Example Usage:

```

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

```

### Backlight\_getLedDimming()

EXTERN\_C CCAUXDLLAPI eErr CCAUXDLL\_CALLING\_CONV CrossControl::Backlight↔  
\_getLedDimming (

```

    BACKLIGHTHANDLE ,
    CCStatus * status )

```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

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

### Backlight\_getStatus()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↔
_getStatus (
    BACKLIGHTHANDLE ,
    uint8_t * status )
```

Get backlight controller status. Deprecated, use Backlight\_getHWStatus instead.

Supported Platform(s): XL, XM

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. See the enum eErr for details.

### Example Usage:

```
err = Backlight_getStatus(pBacklight, &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 if (err == ERR.NOT_SUPPORTED)
{
    printf("Backlight_getStatus: Not supported!\n");
}
else
{
    printf("Error(%d) in function Backlight_getStatus: %s\n", err,
        GetErrorStringA(err));
}
```

### Backlight\_release()

```
EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::Backlight↔
```

```

.release (
    BACKLIGHTHANDLE )

```

Delete the backlight object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### Returns

-

#### Example Usage:

```

BACKLIGHTHANDLE pBacklight = ::GetBacklight();
assert(pBacklight);
if(argc == 2)
{
    change_backlight(pBacklight, (unsigned char)atoi(argv[1]));
}
else
{
    change_backlight(pBacklight, -1);
}
Backlight.release(pBacklight);

```

#### Backlight.setAutomaticBLFilter()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Backlight↔

```

```

.setAutomaticBLFilter (
    BACKLIGHTHANDLE ,
    uint32_t averageWndSize,
    uint32_t rejectWndSize,
    uint32_t rejectDeltaInLux,
    LightSensorSamplingMode mode )

```

Set light sensor filter parameters for automatic backlight control.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

#### 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. See the enum eErr for details.

#### Backlight.setAutomaticBLParams()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Backlight↔

```

```

.setAutomaticBLParams (

```

```
BACKLIGHTHANDLE ,
bool bSoftTransitions )
```

Set parameters for automatic backlight control.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Backlight.setIntensity()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Backlight↔
.setIntensity (
    BACKLIGHTHANDLE ,
    uint8_t intensity )
```

Set backlight intensity. Note that there might be hardware limitations, limiting the minimum and/or maximum value to other than (1..255).

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

<i>intensity</i>	The backlight intensity to set (1..255).
------------------	--

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

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

### Backlight.setLedDimming()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Backlight↔
.setLedDimming (
    BACKLIGHTHANDLE ,
    CCStatus status )
```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Backlight.startAutomaticBL()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↔
.startAutomaticBL (
    BACKLIGHTHANDLE )
```

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

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Backlight.stopAutomaticBL()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↔
.stopAutomaticBL (
    BACKLIGHTHANDLE )
```

Stop automatic backlight control.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.



**GetBacklight()**

```
EXTERN_C CCAUXDLL_API BACKLIGHTHANDLE CCAUXDLL_CALLING_CONV CrossControl↔
::GetBacklight (
    void )
```

Factory function that creates instances of the Backlight object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Returns**

BACKLIGHTHANDLE to an allocated Backlight object. The returned handle needs to be deallocated using the [Backlight\\_release\(BACKLIGHTHANDLE\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

**Example Usage:**

```
BACKLIGHTHANDLE pBacklight = ::GetBacklight();
assert(pBacklight);
if(argc == 2)
{
    change_backlight(pBacklight, (unsigned char)atoi(argv[1]));
}
else
{
    change_backlight(pBacklight, -1);
}
Backlight_release(pBacklight);
```

## 19.5 Battery functions

**Functions**

- EXTERN\_C CCAUXDLL\_API BATTERYHANDLE CCAUXDLL\_CALLING\_CONV↔  
\_CONV [GetBattery](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Battery\\_release](#)  
(BATTERYHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_isBatteryPresent](#)  
(BATTERYHANDLE, bool \*batteryIsPresent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatteryVoltageStatus](#)  
(BATTERYHANDLE, uint8\_t \*batteryVoltagePercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatteryChargingStatus](#)  
(BATTERYHANDLE, ChargingStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getPowerSource](#)  
(BATTERYHANDLE, PowerSource \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatteryTemp](#)  
(BATTERYHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getHwErrorStatus](#)  
(BATTERYHANDLE, ErrorStatus \*errorCode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getTimer](#)  
(BATTERYHANDLE, BatteryTimerType \*times)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getMinMaxTemp](#)  
(BATTERYHANDLE, int16\_t \*minTemp, int16\_t \*maxTemp)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatteryHWversion](#)  
(BATTERYHANDLE, char\_t \*buff, int32\_t len)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getBatterySwVersion (BATTERYHANDLE, uint16\_t \*major, uint16\_t \*minor, uint16\_t \*release, uint16\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getBatterySerial (BATTERYHANDLE, char\_t \*buff, int32\_t len)

## 19.5.1 Detailed Description

Functions in the Battery class

## 19.5.2 Function Documentation

### Battery\_getBatteryChargingStatus()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_get↔
BatteryChargingStatus (
    BATTERYHANDLE ,
    ChargingStatus * status )
```

Get battery charging status.

Supported Platform(s): XM

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
ChargingStatus cs;
error = Battery_getBatteryChargingStatus(pBattery, &cs);
if (error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryChargingStatus: " << GetErrorStringA(error) << " - 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;
    }
}

```

**Battery\_getBatteryHWversion()**

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Battery_get↵
BatteryHWversion (
    BATTERYHANDLE ,
    char_t * buff,
    int32_t len )

```

Get battery hardware version (PCB revision).

Supported Platform(s): XM

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. See the enum eErr for details.

Example Usage:

```

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

```

**Battery\_getBatterySerial()**

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Battery_get↵
BatterySerial (

```

```

    BATTERYHANDLE ,
    char_t * buff,
    int32_t len )

```

Get battery serial number.

Supported Platform(s): XM

#### 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. See the enum eErr for details.

#### Example Usage:

```

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

```

#### Battery\_getBatterySwVersion()

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::Battery.get←  
BatterySwVersion (

```

    BATTERYHANDLE ,
    uint16_t * major,
    uint16_t * minor,
    uint16_t * release,
    uint16_t * build )

```

Get the battery software version

Supported Platform(s): XM

#### 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. See the enum eErr for details.

**Example Usage:**

```

unsigned short major;
unsigned short minor;
unsigned short release;
unsigned short build;
error = Battery.getBatterySwVersion(pBattery, &major, &minor, &release, &build);
if (error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatterySwVersion: " << GetErrorStringA(error) << " - 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;
}

```

**Battery\_getBatteryTemp()**

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Battery_get←
BatteryTemp (
    BATTERYHANDLE ,
    int16_t * temperature )

```

Get battery temperature.

Supported Platform(s): XM

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```

short temp;
error = Battery.getBatteryTemp(pBattery, &temp);
if (error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getBatteryTemp: " << GetErrorStringA(error) << " - 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;
}

```

**Battery\_getBatteryVoltageStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_get↔
BatteryVoltageStatus (
    BATTERYHANDLE ,
    uint8_t * batteryVoltagePercent )
```

Get battery voltage status.

Supported Platform(s): XM

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

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

**Battery\_getHwErrorStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_get↔
HwErrorStatus (
    BATTERYHANDLE ,
    ErrorStatus * errorCode )
```

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

Supported Platform(s): XM

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```

ErrorStatus es;
error = Battery_getHwErrorStatus(pBattery, &es);
if (error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getHwErrorStatus: " << GetErrorStringA(error) << " - 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;
    }
}
}

```

### Battery\_getMinMaxTemp()

EXTERN\_C CCAUXDLLAPI eErr CCAUXDLLCALLING\_CONV CrossControl::Battery\_get↔  
MinMaxTemp (

```

    BATTERYHANDLE ,
    int16_t * minTemp,
    int16_t * maxTemp )

```

Get temperature interval of the battery.

Supported Platform(s): XM

#### Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
short max;
error = Battery.getMinMaxTemp(pBattery, &temp, &max);
if (error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getMinMaxTemp: " << GetErrorStringA(error) << " - 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;
}
```

**Battery\_getPowerSource()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_getPowerSource (
    BATTERYHANDLE ,
    PowerSource * status )
```

Get the currently used power source.  
Supported Platform(s): XM

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
PowerSource ps;
error = Battery.getPowerSource(pBattery, &ps);
if (error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getPowerSource: " << GetErrorStringA(error) << " - 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;
}
```



**Battery\_getTimer()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_getTimer (
    BATTERYHANDLE ,
    BatteryTimerType * times )
```

Get battery diagnostic timer.

Supported Platform(s): XM

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
BatteryTimerType times;
memset(&times, 0, sizeof(times));
error = Battery_getTimer(pBattery, &times);
if (error == ERR_NOT_SUPPORTED && !bpresent)
{
    cout << "getTimer: " << GetErrorStringA(error) << " - 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;
    cout << "Total run time on battery power=" << times.TotRunTimeBattery*60 << "
        min(s)" << std::endl;
    cout << "Total run time below -20C=" << times.RunTime_m20 << " min(s)" <<
        std::endl;
    cout << "Total run time -20-0C=" << times.RunTime_m20_0 << " min(s)" << std::endl;
    cout << "Total run time 0-40C=" << times.RunTime_0_40 << " min(s)" << std::endl;
    cout << "Total run time 40-60C=" << times.RunTime_40_60 << " min(s)" << std::endl;
    cout << "Total run time 60-70C=" << times.RunTime_60_70 << " min(s)" << std::endl;
    cout << "Total run time 70-80C=" << times.RunTime_70_80 << " min(s)" << std::endl;
    cout << "Total run time above 80C=" << times.RunTime_Above80 << " min(s)" <<
        std::endl;
}
}
```

**Battery\_isBatteryPresent()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_isBatteryPresent (
    BATTERYHANDLE ,
    bool * batteryIsPresent )
```

Is an external battery connected?

Supported Platform(s): XM

## Parameters

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

## Returns

-

## Example Usage:

```

error = Battery_isBatteryPresent(pBattery, &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;
    }
}

```

**Battery\_release()**

```

EXTERN_C CCAUXDLLAPI void CCAUXDLLCALLING_CONV CrossControl::Battery_release
(

```

```

    BATTERYHANDLE )

```

Delete the Battery object

Supported Platform(s): XM.

## Returns

-

## Example Usage:

```

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

```

**GetBattery()**

```

EXTERN_C CCAUXDLLAPI BATTERYHANDLE CCAUXDLLCALLING_CONV CrossControl::↵
GetBattery (

```

```

    void )

```

Factory function that creates instances of the Battery object.

Supported Platform(s): XM

**Returns**

BATTERYHANDLE to an allocated battery object. The returned handle needs to be deallocated using the [Battery\\_release\(BATTERYHANDLE\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

**Example Usage:**

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

## 19.6 Buzzer functions

### Functions

- EXTERN\_C CCAUXDLL\_API BuzzerHANDLE CCAUXDLL\_CALLING\_CONV GetBuzzer (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Buzzer\_release (BuzzerHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_getFrequency (BuzzerHANDLE, uint16\_t \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_getVolume (BuzzerHANDLE, uint16\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_getScaledVolume (BuzzerHANDLE, uint8\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_getTrigger (BuzzerHANDLE, bool \*trigger)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_setFrequency (BuzzerHANDLE, uint16\_t frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_setVolume (BuzzerHANDLE, uint16\_t volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_setScaledVolume (BuzzerHANDLE, uint8\_t volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_setTrigger (BuzzerHANDLE, bool trigger)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_buzz (BuzzerHANDLE, int32\_t time, bool blocking)

### 19.6.1 Detailed Description

Functions in the Buzzer class

### 19.6.2 Function Documentation

**Buzzer\_buzze()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Buzzer_buzze
(
    BUZZERHANDLE ,
    int32_t time,
    bool blocking )
```

Buzzes for a specified time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## 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. See the enum eErr for details.

## Example Usage:

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

**Buzzer\_getFrequency()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Buzzer_get←
Frequency (
    BUZZERHANDLE ,
    uint16_t * frequency )
```

Get buzzer frequency.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Buzzer\_getScaledVolume()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Buzzer_get←
ScaledVolume (
    BUZZERHANDLE ,
    uint8_t * volume )
```

Get scaled buzzer volume.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

<i>volume</i>	Current volume 0-100%
---------------	-----------------------

Due limitation in HW, readback value may differ slightly from the value that was set.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

**Buzzer\_getTrigger()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Buzzer_get←
Trigger (
    BUZZERHANDLE ,
    bool * trigger )
```

Get buzzer trigger. The Buzzer is enabled when the trigger is enabled.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Buzzer\_getVolume()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Buzzer_get←
Volume (
    BUZZERHANDLE ,
    uint16_t * volume )
```

Get buzzer volume. Note: For platform compatibility, use `Buzzer_getScaledVolume` instead. This function may be removed in a future API version. Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

<i>volume</i>	Current volume (0-51) (VS: 0-2000) (VI2: 0-4095).
---------------	---

Returns

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

Example Usage:

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

### Buzzer\_release()

```
EXTERN_C CCAUXDLLAPI void CCAUXDLL_CALLING_CONV CrossControl::Buzzer_release
(
```

```
    BUZZERHANDLE )
```

Delete the Buzzer object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Returns

-

Example Usage:

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

### Buzzer\_setFrequency()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Buzzer_set←
Frequency (
```

```
    BUZZERHANDLE ,
    uint16_t frequency )
```

Set buzzer frequency.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

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

## Returns

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

## Example Usage:

```
err = Buzzer.setFrequency(pBuzzer, freq);
if (err != ERR.SUCCESS)
{
    cout << "Error(" << err << ") in function setFrequency: " <<
        GetErrorStringA(err) << endl;
}
else
{
    err = Buzzer.buzze(pBuzzer, duration, true);
    if (err != ERR.SUCCESS)
    {
        cout << "Error(" << err << ") in function buzze: " << GetErrorStringA(err)
            << endl;
    }
}
```

**Buzzer.setScaledVolume()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Buzzer.set←
ScaledVolume (
    BUZZERHANDLE ,
    uint8_t volume )
```

Set scaled buzzer volume.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

<i>volume</i>	Volume to set 0-100%.
---------------	-----------------------

## Returns

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

## Example Usage:

**Buzzer.setTrigger()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Buzzer.set←
Trigger (
    BUZZERHANDLE ,
    bool trigger )
```

Set buzzer trigger. The Buzzer is enabled when the trigger is enabled.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

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

## Returns

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

**Buzzer\_setVolume()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Buzzer_setVolume (
    BUZZERHANDLE ,
    uint16_t volume )
```

Set buzzer volume. Note: For platform compatibility, use Buzzer\_setScaledVolume instead. This function may be removed in a future API version.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

<i>volume</i>	Volume to set (0-51) (VS: 0-2000) (VI2: 0-4095) (V700: 0-50)
---------------	--

## Returns

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

## Example Usage:

```
err = Buzzer_setVolume( pBuzzer, 20);
if (err == ERR.SUCCESS)
{
    cout << "Buzzer volume set to 20" << endl;
}
else
{
    cout << "Error(" << err << ") in function setVolume: " << GetErrorStringA(err)
        << endl;
}
```

**GetBuzzer()**

```
EXTERN_C CCAUXDLLAPI BUZZERHANDLE CCAUXDLLCALLING_CONV CrossControl::GetBuzzer (
    void )
```

Factory function that creates instances of the Buzzer object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700



**Returns**

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

**Example Usage:**

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

## 19.7 CanSetting functions

**Functions**

- EXTERN\_C CCAUXDLL\_API [CANSETTINGHANDLE](#) CCAUXDLL\_CALLING↔\_CONV [GetCanSetting](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [CanSetting\\_release](#) ([CANSETTINGHANDLE](#))
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [CanSetting\\_getBaudrate](#) ([CANSETTINGHANDLE](#), [uint8\\_t](#) net, [uint16\\_t](#) \*baudrate)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [CanSetting\\_getFrameType](#) ([CANSETTINGHANDLE](#), [uint8\\_t](#) net, [CanFrameType](#) \*frameType)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [CanSetting\\_setBaudrate](#) ([CANSETTINGHANDLE](#), [uint8\\_t](#) net, [uint16\\_t](#) baudrate)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [CanSetting\\_setFrameType](#) ([CANSETTINGHANDLE](#), [uint8\\_t](#) net, [CanFrameType](#) frameType)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [CanSetting\\_getFlexibleDataBaudrates](#) ([CANSETTINGHANDLE](#), [uint8\\_t](#) net, [uint16\\_t](#) \*baudrateNominal, [uint16\\_t](#) \*baudrate↔Data)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [CanSetting\\_setFlexibleDataBaudrates](#) ([CANSETTINGHANDLE](#), [uint8\\_t](#) net, [uint16\\_t](#) baudrateNominal, [uint16\\_t](#) baudrate↔Data)

### 19.7.1 Detailed Description

Functions in the CanSetting class

### 19.7.2 Function Documentation

**CanSetting\_getBaudrate()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CanSetting↔_getBaudrate (
    CANSETTINGHANDLE ,
    uint8\_t net,
    uint16\_t * baudrate )
```

Get Baud rate

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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. See the enum eErr for details.

Example Usage:

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

**CanSetting\_getFlexibleDataBaudrates()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CanSetting↔
_getFlexibleDataBaudrates (
    CANSETTINGHANDLE ,
    uint8_t net,
    uint16_t * baudrateNominal,
    uint16_t * baudrateData )
```

Get FD Baudrates

Supported Platform(s): XL5, XM9

Parameters

<i>net</i>	CAN net (1-4) to get settings for.
<i>baudrateNominal</i>	CAN baudrateNominal selection (kbit/s).
<i>baudrateData</i>	CAN baudrateData selection (kbit/s). If the returned baudrates are ZERO then settings are such that FD-FLEXIBLE_DATA is not active a CLASSIC baudrate is active and the API <a href="#">CanSetting_getBaudrate()</a> should be used.

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

**CanSetting\_getFrameType()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CanSetting↔
_getFrameType (
```

```

CANSETTINGHANDLE ,
uint8_t net,
CanFrameType * frameType )

```

Get frame type

Supported Platform(s): XL, XL5, XM, XM9

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. See the enum eErr for details.

Example Usage:

```

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

```

### CanSetting\_release()

```

EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::CanSetting↔
_release (

```

```

    CANSETTINGHANDLE )

```

Delete the CanSetting object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Returns

-

Example Usage:

```

CANSETTINGHANDLE pCanSetting = ::GetCanSetting();
assert(pCanSetting);
read_cansettings(pCanSetting);
CanSetting_release(pCanSetting);

```

### CanSetting\_setBaudrate()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CanSetting↔
_setBaudrate (

```

```

    CANSETTINGHANDLE ,
    uint8_t net,
    uint16_t baudrate )

```

Set Baud rate. The changes will take effect after a restart (Windows). The changes will be reset after a restart (Linux)

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## 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. See the enum eErr for details.

**CanSetting\_setFlexibleDataBaudrates()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CanSetting↔
_setFlexibleDataBaudrates (
    CANSETTINGHANDLE ,
    uint8_t net,
    uint16_t baudrateNominal,
    uint16_t baudrateData )
```

Set FD Baudrates.

Supported Platform(s): XL5, XM9

## Parameters

<i>net</i>	CAN net (1-4).
<i>baudrateNominal</i>	CAN baudrateNominal selection (kbit/s).
<i>baudrateData</i>	CAN baudrateData selection (kbit/s).

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

**CanSetting\_setFrameType()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CanSetting↔
_setFrameType (
    CANSETTINGHANDLE ,
    uint8_t net,
    CanFrameType frameType )
```

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

Supported Platform(s): XL, XL5, XM, XM9

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**GetCanSetting()**

```
EXTERN_C CCAUXDLL_API CANSETTINGHANDLE CCAUXDLL_CALLING_CONV CrossControl↔
::GetCanSetting (
    void )
```

Factory function that creates instances of the CanSetting object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Returns

CANSETTINGHANDLE to an allocated CanSetting object. The returned handle needs to be deallocated using the [CanSetting\\_release\(CANSETTINGHANDLE\)](#) 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);
CanSetting_release(pCanSetting);
```

## 19.8 CfgIn functions

### Functions

- EXTERN\_C CCAUXDLL\_API CFGINHANDLE CCAUXDLL\_CALLING\_CONV↔  
GetCfgIn (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV CfgIn\_release  
(CFGINHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_setCfgInMode  
(CFGINHANDLE, uint8\_t channel, CfgInModeEnum set\_mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getCfgInMode  
(CFGINHANDLE, uint8\_t channel, CfgInModeEnum \*get\_mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getValue  
(CFGINHANDLE, uint8\_t channel, uint16\_t \*sample\_value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getPwmValue  
(CFGINHANDLE, uint8\_t channel, float32\_t \*frequency, uint8\_t \*duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getFrequencyValue  
(CFGINHANDLE, uint8\_t channel, float32\_t \*frequency)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getMinFrequencyThreshold (CFGINHANDLE, uint8\_t channel, float32\_t \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_setMinFrequencyThreshold (CFGINHANDLE, uint8\_t channel, float32\_t frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_setFrequencyFilterLevel (CFGINHANDLE, uint8\_t level)

## 19.8.1 Detailed Description

Functions in the CfgIn class

## 19.8.2 Function Documentation

### CfgIn\_getCfgInMode()

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::CfgIn\_getCfgInMode (

CFGINHANDLE ,  
uint8\_t channel,  
CfgInModeEnum \* get\_mode )

Get Configurable Input mode

Supported Platform(s): VC, VA, VS, VI2

Parameters

<i>channel</i>	Which configurable input channel to use, 1-2 (VC), 1-8 (VA), 1 (VS) or 1-4 (VI2) corresponding to physical input channel
<i>get_mode</i>	Storage container for retrieved mode Configurable input can be set to different measurement modes, this reads the setting back

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
err = CfgIn_getCfgInMode(pCfgIn, channel, &get_mode);
if (err != ERR_SUCCESS)
{
    cout << "CfgIn_getCfgInMode: " << GetErrorStringA(err) << std::endl;
}
else
{
    switch(get_mode)
    {
        case CFGIN_NOT_IN_USE: cout << "CfgIn_getCfgInMode (" << (int)channel << "):
        CFGIN_NOT_IN_USE" << std::endl; break;
        case CFGIN_HI_SWITCH: cout << "CfgIn_getCfgInMode (" << (int)channel << "):
        CFGIN_HI_SWITCH" << std::endl; break;
        case CFGIN_LOW_SWITCH: cout << "CfgIn_getCfgInMode (" << (int)channel << "):
        CFGIN_LOW_SWITCH" << std::endl; break;
        case CFGIN_VOLTAGE_2V5: cout << "CfgIn_getCfgInMode (" << (int)channel << "):
        CFGIN_VOLTAGE_2V5" << std::endl; break;
```

```

case CFGIN_VOLTAGE_5V: cout << "CfgIn.getCfgInMode (" << (int)channel << "):
CFGIN_VOLTAGE_5V" << std::endl; break;
case CFGIN_RESISTANCE: cout << "CfgIn.getCfgInMode (" << (int)channel << "):
CFGIN_RESISTANCE" << std::endl; break;
case CFGIN_FREQ_FLOATING: cout << "CfgIn.getCfgInMode (" << (int)channel <<
"): CFGIN_FREQ_FLOATING" << std::endl; break;
case CFGIN_FREQ_PULLUP: cout << "CfgIn.getCfgInMode (" << (int)channel << "):
CFGIN_FREQ_PULLUP" << std::endl; break;
case CFGIN_FREQ_PULLDOWN: cout << "CfgIn.getCfgInMode (" << (int)channel <<
"): CFGIN_FREQ_PULLDOWN" << std::endl; break;
default: cout << "CfgIn.getCfgInMode (" << (int)channel << "): Unknown mode"
<< std::endl; break;
}
}

```

**CfgIn.getFrequencyValue()**

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::CfgIn.getFrequency↔
Value (
    CFGINHANDLE ,
    uint8_t channel,
    float32_t * frequency )

```

Read the sampled frequency value from configurable input, when in frequency mode:

VA: For ports 1-4, time base is 72 MHz ± 100 ppm (more accurate) For ports 5-8, time base is 60 kHz ± 100 ppm (less accurate) Input range is 0 Hz – 15 kHz. See technical manual for more details.

VA: For all ports 1-8: CFGIN\_FREQ\_PD\_5V - sample\_value in Hz CFGIN\_FREQ\_PD\_10V - sample\_value in Hz CFGIN\_FREQ\_PD\_32V - sample\_value in Hz CFGIN\_FREQ\_F\_5V - sample\_value in Hz CFGIN\_FREQ\_F\_10V - sample\_value in Hz CFGIN\_FREQ\_F\_32V - sample\_value in Hz

VA: For ports 5-8 only: CFGIN\_FREQ\_PU\_5V - sample\_value in Hz CFGIN\_FREQ\_PU\_10V - sample\_value in Hz CFGIN\_FREQ\_PU\_32V - sample\_value in Hz

VS: Port 1 only - sample\_value in Hz

VI2: For ports 1-4 time base is 48 Mz ±100 ppm Input range is 0.1 Hz - 50 kHz. See technical manual for more details.

VI2: For all ports 1-4: CFGIN\_FREQ\_FLOATING - sample\_value in Hz CFGIN\_FREQ\_PULLUP - sample\_value in Hz CFGIN\_FREQ\_PULLDOWN - sample\_value in Hz

Supported Platform(s): VA, VS, VI2

Parameters

<i>channel</i>	Which configurable input channel to use, 1 through 8, corresponding to physical input channel
<i>frequency</i>	Read signal frequency in Hz; signal resolution and range depending on mode and port

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

**CfgIn\_getMinFrequencyThreshold()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL-CALLING-CONV CrossControl::CfgIn-getMinFrequencyThreshold (
    CFGINHANDLE ,
    uint8_t channel,
    float32_t * frequency )
```

Read the configured minimum frequency threshold for configurable input, when in frequency measurement mode (CFGIN\_FREQ\_FLOATING, CFGIN\_FREQ\_PULLUP, CFGIN\_FREQ\_PULLDOWN for VC and VI2, and CFGIN\_FREQ\_PD\_5V, CFGIN\_FREQ\_PD\_10V, CFGIN\_FREQ\_PD\_32V, CFGIN\_FREQ\_F\_5V, CFGIN\_FREQ\_F\_10V, CFGIN\_FREQ\_F\_32V, CFGIN\_FREQ\_PU\_5V, CFGIN\_FREQ\_PU\_10V, CFGIN\_FREQ\_PU\_32V for VA). The frequency threshold is set to 1Hz at device start-up. Use the frequency threshold to set up how fast to detect a frequency change or a static signal. If you know the frequency range of the measured signal - set the threshold slightly lower than this. That way, a change from pulses to a static signal is detected as fast as possible. If the frequency threshold is set to e.g. 0.1Hz, it can take up to 10 seconds before a change in frequency is detected - also depending on the actual frequency of the signal. For VC, when the measured signal is slower than the frequency threshold, CfgIn\_getPwmValue will return frequency 0Hz, duty cycle 0 or 100%. For VA, when the measured signal is slower than the frequency threshold, CfgIn\_getFrequencyValue will return frequency 0 Hz. For VS, this function/setting only applies for input channel 1.

Supported Platform(s): VC, VA, VS, VI2

## Parameters

<i>channel</i>	Which configurable input channel to use, 1-2 (VC), 1-8 (VA), 1 (VS) or 1-4 (VI2), corresponding to physical input channel
<i>frequency</i>	Minimum frequency threshold, 0.0 - 50000.0 Hz for VC and VS, 0 - 15000 Hz for VA.

## Returns

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

## Example Usage:

```
float frequency_threshold;
channel = 2;
err = CfgIn_getMinFrequencyThreshold(pCfgIn, channel, &frequency_threshold);
if (err != ERR.SUCCESS)
{
    cout << "CfgIn_getMinFrequencyThreshold: " << GetErrorStringA(err) <<
        std::endl;
}
else
{
    cout << "CfgIn_getMinFrequencyThreshold: channel 2: " << std::fixed <<
        frequency_threshold << "Hz" << std::endl;
}
```



**CfgIn\_getPwmValue()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CfgIn_getPwmValue (
    CFGINHANDLE ,
    uint8_t channel,
    float32_t * frequency,
    uint8_t * duty_cycle )
```

Read the sampled value from configurable input, when in frequency measurement mode (CFGIN\_FREQ\_FLOATING, CFGIN\_FREQ\_PULLUP, CFGIN\_FREQ\_PULLDOWN).

Supported Platform(s): VC, VA

## Parameters

<i>channel</i>	Which configurable input channel to use, 1 or 2, corresponding to physical input channel
<i>frequency</i>	Read signal frequency, 0.0 - 50000.0 Hz
<i>duty_cycle</i>	Read signal duty cycle, 0-100%

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
float frequency;
unsigned char duty_cycle;
err = CfgIn_getPwmValue(pCfgIn, 2, &frequency, &duty_cycle);
if (err != ERR_SUCCESS)
{
    cout << "CfgIn_getPwmValue: " << GetErrorStringA(err) << std::endl;
}
else
{
    cout << "CfgIn_getPwmValue: channel 2 PWM measurement: " << std::fixed <<
        frequency << "Hz, " << (int)duty_cycle << "% duty cycle" << std::endl;
}
```

**CfgIn\_getValue()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CfgIn_getValue (
    CFGINHANDLE ,
    uint8_t channel,
    uint16_t * sample_value )
```

Read the sampled value from configurable input, when in modes other than frequency mode:

For VC platform: CFGIN\_HI\_SWITCH - sample\_value is 0-1 CFGIN\_LOW\_SWITCH - sample\_value is 0-1 CFGIN\_VOLTAGE\_2V5 - sample\_value is 0-30000 (0.1mV steps) CFGIN\_VOLTAGE\_5V - sample\_value is 0-60000 (0.1mV steps) CFGIN\_RESISTANCE - sample\_value is 0-65535 Ohm

For VA platform, all ports 1-8: CFGIN\_CURRENT\_4\_20 - sample\_value in  $\mu\text{A}$ , accuracy  $\pm 0.5\% \pm 50 \mu\text{A}$  (typical) CFGIN\_VOLTAGE\_5V - sample\_value in mV, accuracy  $\pm 0.5\% \pm 5 \text{ mV}$  (typical) CFGIN\_VOLTAGE\_10V - sample\_value in mV, accuracy  $\pm 0.5\% \pm 10 \text{ mV}$  (typical) CFGIN\_VOLTAGE\_32V - sample\_value in mV, accuracy  $\pm 0.5\% \pm 32 \text{ mV}$  (typical) CFGIN\_DIGITAL\_PD\_5V - sample\_value is 0-1 CFGIN\_DIGITAL\_PD\_10V - sample\_value is 0-1 CFGIN\_DIGITAL\_PD\_32V - sample\_value is 0-1 CFGIN\_DIGITAL\_F\_5V - sample\_value is 0-1 CFGIN\_DIGITAL\_F\_10V - sample\_value is 0-1 CFGIN\_DIGITAL\_F\_32V - sample\_value is 0-1

For VA platform, ports 1-4 only: CFGIN\_RESISTANCE - sample\_value in Ohm, accuracy  $\pm 0.5\% \pm 5 \text{ Ohm}$  (typical) CFGIN\_RESISTANCE\_500 - sample\_value in 0.1 Ohm/bit, accuracy  $\pm 0.5\% \pm 0.5 \text{ Ohm}$  (typical)

For VA platform, ports 5-8 only: CFGIN\_DIGITAL\_PU\_5V - sample\_value is 0-1 CFGIN\_DIGITAL\_PU\_10V - sample\_value is 0-1 CFGIN\_DIGITAL\_PU\_32V - sample\_value is 0-1

For VS platform, input 1-2 in any mode: sample\_value in mV.

For VI2 platform, all ports 1-4: CFGIN\_HI\_SWITCH - sample\_value is 0-1 CFGIN\_LOW\_SWITCH - sample\_value is 0-1 CFGIN\_VOLTAGE\_5V - sample\_value is 0-5500 (1mV steps) CFGIN\_RESISTANCE - sample\_value is 0-500 (0.01 Ohm steps)

Supported Platform(s): VC, VA, VS, VI2

#### Parameters

<i>channel</i>	Which configurable input channel to use, 1-2 (VC), 1-8 (VA), 1-2 (VS) or 1-4 (VI2), corresponding to physical input channel
<i>sample_value</i>	Read value which is relevant to actual mode setting The actual value is dependent on the mode setting

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

#### Example Usage:

```
unsigned short value;
err = CfgIn_getValue(pCfgIn, 1, &value);
if (err != ERR_SUCCESS)
{
    cout << "CfgIn_getValue: " << GetErrorStringA(err) << std::endl;
}
else
{
    cout << "CfgIn_getValue: channel 1 2V5 voltage measurement: " << (int)value <<
        "mV" << std::endl;
}
```

#### CfgIn\_release()

```
EXTERN_C CCAUXDLLAPI void CCAUXDLL_CALLING_CONV CrossControl::CfgIn_release
(
    CFGINHANDLE )
```

Delete the CfgIn object.

Supported Platform(s): VC, VA, VS, VI2

Returns

-

Example Usage:

```
CFGINHANDLE pCfgIn = ::GetCfgIn();
assert(pCfgIn);
cfgin_example(pCfgIn);
CfgIn.release(pCfgIn);
```

**CfgIn.setCfgInMode()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::CfgIn.setCfgInMode(
    InMode (
```

```
        CFGINHANDLE ,
        uint8_t channel,
        CfgInModeEnum set_mode )
```

Set Configurable Input mode  
Supported Platform(s): VC, VA, VS, VI2

Parameters

<i>channel</i>	Which configurable input channel to use, 1-2 (VC), 1-8 (VA), 1 (VS) or 1-4 (VI2), corresponding to physical input channel
<i>set_mode</i>	Which mode to set Configurable input can be set to different measurement modes. See CfgInModeEnum for a description of which platform and input combinations are possible.

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
err = CfgIn.setCfgInMode(pCfgIn, channel, CFGIN_VOLTAGE_2V5);
if (err != ERR_SUCCESS)
{
    cout << "CfgIn.setCfgInMode: " << GetErrorStringA(err) << std::endl;
}
else
{
    cout << "CfgIn.setCfgInMode: channel 1 mode set to CFGIN_VOLTAGE_2V5" <<
        std::endl;
}
```

**CfgIn.setFrequencyFilterLevel()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::CfgIn.setFrequencyFilterLevel(
    FilterLevel (
```

```
        CFGINHANDLE ,
        uint8_t level )
```

Sets the weight of the old sample value in frequency measurements as a percentage. The sampled frequency is filtered with a moving average. A large weight increases the

filter level and gives better accuracy in high frequency measurements, but decreases the speed of which changes in the input frequency can be detected.

Supported Platform(s): VC, VA

Parameters

<i>level</i>	Weight of the old sample value as a percentage (0-99)
--------------	---

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

**CfgIn\_setMinFrequencyThreshold()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::CfgIn_setMinFrequencyThreshold (
    CFGINHANDLE ,
    uint8_t channel,
    float32_t frequency )
```

Set the minimum frequency threshold for configurable input, when in frequency measurement mode (CFGIN\_FREQ\_FLOATING, CFGIN\_FREQ\_PULLUP, CFGIN\_FREQ\_PULLDOWN for VC and VI2, and CFGIN\_FREQ\_PD\_5V, CFGIN\_FREQ\_PD\_10V, CFGIN\_FREQ\_PD\_32V, CFGIN\_FREQ\_F\_5V, CFGIN\_FREQ\_F\_10V, CFGIN\_FREQ\_F\_32V, CFGIN\_FREQ\_PU\_5V, CFGIN\_FREQ\_PU\_10V, CFGIN\_FREQ\_PU\_32V for VA). The frequency threshold is set to 1Hz at device start-up. Use the frequency threshold to set up how fast to detect a frequency change or a static signal. If you know the frequency range of the measured signal - set the threshold slightly lower than this. That way, a change from pulses to a static signal is detected as fast as possible. If the frequency threshold is set to e.g. 0.1Hz, it can take up to 10 seconds before a change in frequency is detected - also depending on the actual frequency of the signal. For VC, when the measured signal is slower than the frequency threshold, CfgIn\_getPwmValue will return frequency 0Hz, duty cycle 0 or 100%. For VA, when the measured signal is slower than the frequency threshold, CfgIn\_getFrequencyValue will return frequency 0 Hz. For VS, this function/setting only applies for input channel 1.

Supported Platform(s): VC, VA, VS, VI2

Parameters

<i>channel</i>	Which configurable input channel to use, 1-2 (VC), 1-8 (VA), 1 (VS) or 1-4 (VI2), corresponding to physical input channel
<i>frequency</i>	Minimum frequency threshold, 0.0 - 50000.0 Hz for VC, VS and VI2, 0-15000 Hz for VA.

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
channel = 2;
err = CfgIn.setMinFrequencyThreshold(pCfgIn, channel, 50.0);
if (err != ERR_SUCCESS)
{
    cout << "CfgIn.setMinFrequencyThreshold: " << GetErrorStringA(err) <<
        std::endl;
}
else
{
    cout << "CfgIn.setMinFrequencyThreshold: channel 2 minimum frequency threshold
        set to 50.0Hz" << std::endl;
}
```

**GetCfgIn()**

```
EXTERN_C CCAUXDLL_API CFGINHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get↔
CfgIn (
```

```
void )
```

Factory function that creates instances of the CfgIn object.

Supported Platform(s): VC, VA, VS, VI2

**Returns**

CFGINHANDLE to an allocated CfgIn object. The returned handle needs to be deallocated using the [CfgIn.release\(CFGINHANDLE\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

**Example Usage:**

```
CFGINHANDLE pCfgIn = ::GetCfgIn();
assert(pCfgIn);
cfgin.example(pCfgIn);
CfgIn.release(pCfgIn);
```

## 19.9 Config functions

**Functions**

- EXTERN\_C CCAUXDLL\_API CONFIGHANDLE CCAUXDLL\_CALLING\_CONV↔  
\_CONV [GetConfig](#) ()
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Config.release](#)  
(CONFIGHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config.getStartupTriggerConfig](#)  
(CONFIGHANDLE, TriggerConf \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config.getShortButtonPressAction](#)  
(CONFIGHANDLE, PowerAction \*action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config.getLongButtonPressAction](#)  
(CONFIGHANDLE, PowerAction \*action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config.getOnOffSigAction](#)  
(CONFIGHANDLE, PowerAction \*action)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getFrontBtnTrigTime (CONFIGHANDLE, uint16\_t \*triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getExtOnOffSigTrigTime (CONFIGHANDLE, uint32\_t \*triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getButtonFunction (CONFIGHANDLE, uint8\_t button\_number, ButtonConfigEnum \*button\_config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getSuspendMaxTime (CONFIGHANDLE, uint16\_t \*maxTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getCanStartupPowerConfig (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getVideoStartupPowerConfig (CONFIGHANDLE, uint8\_t \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getExtFanStartupPowerConfig (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getStartupVoltageConfig (CONFIGHANDLE, float64\_t \*voltage)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getHeatingTempLimit (CONFIGHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getPowerOnStartup (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setStartupTriggerConfig (CONFIGHANDLE, TriggerConf conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setShortButtonPressAction (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setLongButtonPressAction (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setOnOffSigAction (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setFrontBtnTrigTime (CONFIGHANDLE, uint16\_t triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setExtOnOffSigTrigTime (CONFIGHANDLE, uint32\_t triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setButtonFunction (CONFIGHANDLE, uint8\_t button\_number, ButtonConfigEnum button\_config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setSuspendMaxTime (CONFIGHANDLE, uint16\_t maxTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setCanStartupPowerConfig (CONFIGHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setVideoStartupPowerConfig (CONFIGHANDLE, uint8\_t config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setExtFanStartupPowerConfig (CONFIGHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setStartupVoltageConfig (CONFIGHANDLE, float64\_t voltage)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setHeatingTempLimit (CONFIGHANDLE, int16\_t temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setPowerOnStartup (CONFIGHANDLE, CCStatus status)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setRS485Enabled (CONFIGHANDLE, RS4XXPort port, bool enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getRS485Enabled (CONFIGHANDLE, RS4XXPort port, bool \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setOnOffTriggerMode (CONFIGHANDLE, ConfigOnOffTriggerMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getOnOffTriggerMode (CONFIGHANDLE, ConfigOnOffTriggerMode \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setNextBootMode (CONFIGHANDLE, BootModeEnum mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getNextBootMode (CONFIGHANDLE, BootModeEnum \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setOSAliveMonitoring (CONFIGHANDLE, CCStatus enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getOSAliveMonitoring (CONFIGHANDLE, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getOnOffSignalState (CONFIGHANDLE, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getDigPowerOutputStartupConfig (CONFIGHANDLE, PowerOutput output, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setDigPowerOutputStartupConfig (CONFIGHANDLE, PowerOutput output, CCStatus enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getShortBeepSettings (CONFIGHANDLE, uint16\_t \*duration, uint16\_t \*frequency, uint16\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setShortBeepSettings (CONFIGHANDLE, uint16\_t duration, uint16\_t frequency, uint16\_t volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getLongBeepSettings (CONFIGHANDLE, uint16\_t \*duration, uint16\_t \*frequency, uint16\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setLongBeepSettings (CONFIGHANDLE, uint16\_t duration, uint16\_t frequency, uint16\_t volume)

### 19.9.1 Detailed Description

Video channel 4 config

Functions in the Config class

### 19.9.2 Function Documentation

#### Config\_getButtonFunction()

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::Config\_get↔

```
ButtonFunction (
    CONFIGHANDLE ,
    uint8_t button_number,
    ButtonConfigEnum * button_config )
```

Get Button Function Configuration

Supported Platform(s): VC, VA, VS, VI2

## Parameters

<i>button_number</i>	Which button to configure (1-MAX_BUTTONS)
<i>button_config</i>	Bitfield for button configuration, see enum ButtonConfigEnum for details.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```

CrossControl::ButtonConfigEnum btnconf;
CrossControl::eErr error;
for (int i = 1; i < 9; i++)
{
    error = Config_getButtonFunction(pConfig, i, &btnconf);
    if (error != ERR_SUCCESS)
    {
        cout << "Error(" << error << ") in function Config_getButtonFunction: " <<
            GetErrorStringA(error) << std::endl;
    }
    else
    {
        cout << "Button " << (int)i << " is set to: ";
        switch(btnconf)
        {
            case BUTTON_ONLY_MP_ACTION: cout << "Application only" << std::endl; break;
            case BUTTON_AS_STARTUP_TRIG: cout << "Startup trigger" << std::endl; break;
            case BUTTON_AS_ACTION_TRIG: cout << "Action trigger" << std::endl; break;
            case BUTTON_AS_ACTION_STARTUP_TRIG: cout << "Action and Startup trigger" <<
                std::endl; break;
            case BUTTON_AS_BACKLIGHT_DECREASE: cout << "Backlight decrease" << std::endl;
                break;
            case BUTTON_AS_BACKLIGHT_DECR_STARTUP_TRIG: cout << "Backlight decrease and
                Startup trigger" << std::endl; break;
            case BUTTON_AS_BACKLIGHT_INCREASE: cout << "Backlight increase" << std::endl;
                break;
            case BUTTON_AS_BACKLIGHT_INCR_STARTUP_TRIG: cout << "Backlight increase and
                Startup trigger" << std::endl; break;
            default: cout << "Invalid value" << std::endl; break;
        }
    }
}

```

**Config\_getCanStartupPowerConfig()**

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get←
CanStartupPowerConfig (
    CONFIGHANDLE ,
    CCStatus * status )

```

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.

Supported Platform(s): XL, XL5, XM, XS, XA

## Parameters

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



## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_getDigPowerOutputStartupConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get←
DigPowerOutputStartupConfig (
    CONFIGHANDLE ,
    PowerOutput output,
    CCStatus * enabled )
```

Get power output startup configuration. Power outputs 1 and 2 can be set to be turned on automatically at system startup.

Supported Platform(s): VS

## Parameters

<i>output</i>	Select power output (PowerOutput1 or PowerOutput2) .
<i>enabled</i>	Is the power output configured to be enabled at startup?

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_getExtFanStartupPowerConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get←
ExtFanStartupPowerConfig (
    CONFIGHANDLE ,
    CCStatus * status )
```

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.

Supported Platform(s): XL, XL5, XM

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_getExtOnOffSigTrigTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get←
ExtOnOffSigTrigTime (
    CONFIGHANDLE ,
    uint32_t * triggertime )
```

Get external on/off signal trigger time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## 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. See the enum eErr for details.

**Config\_getFrontBtnTrigTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get←
FrontBtnTrigTime (
    CONFIGHANDLE ,
    uint16_t * triggertime )
```

Get front button trigger time for long press.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2

## 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. See the enum eErr for details.

**Config\_getHeatingTempLimit()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get←
HeatingTempLimit (
    CONFIGHANDLE ,
    int16_t * temperature )
```

Get the temperature 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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_getLongBeepSettings()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_get←
LongBeepSettings (
    CONFIGHANDLE ,
    uint16_t * duration,
    uint16_t * frequency,
    uint16_t * volume )
```

Get Long Beep Settings The computer issues a long beep at certain system events, for example a long button press. This beep may be configured or disabled by setting any of the parameters to 0.

Supported Platform(s): VS, V700

Parameters

<i>duration</i>	Long beep duration (ms)
<i>frequency</i>	Long beep frequency (Hz)
<i>volume</i>	Long beep volume (0-2000)

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_getLongButtonPressAction()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_get←
LongButtonPressAction (
    CONFIGHANDLE ,
    PowerAction * action )
```

Get long button press action. Gets the configured action for a long button press: NoAction, ActionSuspend or ActionShutDown. A long button press is determined by the FrontBtnTrigTime.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_getNextBootMode()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_getNextBootMode (
    CONFIGHANDLE ,
    BootModeEnum * mode )
```

Get BootMode for next boot

Supported Platform(s): VS, VI2, V700

Parameters

<i>mode</i>	Next BootMode
-------------	---------------

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_getOnOffSigAction()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_getOnOffSigAction (
    CONFIGHANDLE ,
    PowerAction * action )
```

Get On/Off signal action. Gets the configured action for an On/Off signal event: NoAction, ActionSuspend or ActionShutDown. An On/Off signal event is determined by the ExtOnOffSigTrigTime.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_getOnOffSignalState()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_getOnOffSignalState (
```

```
    CONFIGHANDLE ,  
    CCStatus * enabled )
```

Get OnOff signal state

Supported Platform(s): XM9, XL5, XA, XS, VC, VA, VS, VI2, V700

**Parameters**

<i>enabled</i>	Is OnOff signal enabled/disabled
----------------	----------------------------------

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_getOnOffTriggerMode()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_getOnOffTriggerMode (
```

```
    CONFIGHANDLE ,  
    ConfigOnOffTriggerMode * mode )
```

Get OnOff/Ignition/KeySwitch signal trigger mode.

Supported Platform(s): XM9, XL5, XA, XS, VC, VA, VS, VI2, V700

**Parameters**

<i>mode</i>	Signal trigger mode. See ConfigOnOffTriggerMode for details
-------------	---

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_getOSAliveMonitoring()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_getOSAliveMonitoring (
```

```
CONFIGHANDLE ,
CCStatus * enabled )
```

Get OS Alive Monitoring

Supported Platform(s): VS, VI2, V700

Parameters

<i>enabled</i>	Is OS Alive monitoring enabled/disabled
----------------	---

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_getPowerOnStartup()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_get↵
PowerOnStartup (
    CONFIGHANDLE ,
    CCStatus * status )
```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_getRS485Enabled()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_get↵
RS485Enabled (
    CONFIGHANDLE ,
    RS4XXPort port,
    bool * enabled )
```

Get RS485 mode configuration for RS4XX port.

Supported Platform(s): XA, XS

## Parameters

<i>port</i>	RS4XX port (RS4XXPort1-4)
<i>enabled</i>	Is the RS485 port enabled (true/false)

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_getShortBeepSettings()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_getShortBeepSettings (
    CONFIGHANDLE ,
    uint16_t * duration,
    uint16_t * frequency,
    uint16_t * volume )
```

Get Short Beep Settings The computer issues a short beep at certain system events such as boot or button press. This beep may be configured or disabled by setting any of the parameters to 0.

Supported Platform(s): VS, V700

## Parameters

<i>duration</i>	Short beep duration (ms)
<i>frequency</i>	Short beep frequency (Hz)
<i>volume</i>	Short beep volume (0-2000)

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_getShortButtonPressAction()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_getShortButtonPressAction (
    CONFIGHANDLE ,
    PowerAction * action )
```

Get short button press action. Gets the configured action for a short button press: NoAction, ActionSuspend or ActionShutDown.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V12

## Parameters

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

## Returns

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

**Config\_getStartupTriggerConfig()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_get←
StartupTriggerConfig (
    CONFIGHANDLE ,
    TriggerConf * config )
```

Get Start-up trigger configuration. Is the front button and/or the external on/off (ignition) signal enabled as triggers for startup and wake up from suspended mode? VC platform: CI state change and Can activity also available as wakeup triggers from suspend mode. See enum TriggerConf for more details.

Supported Platform(s): XL, XM, XL5, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

<i>config</i>	See enum TriggerConf.
---------------	-----------------------

## Returns

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

## Example Usage:

```
err = Config_getStartupTriggerConfig(pConfig, &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;
        // The below values are only available on the VC platform
        case CAN.Button.Activity: cout << "Wake up on CAN and Buttons" << endl; break;
        case CAN.OnOff.Activity: cout << "Wake up on CAN and On/Off/Ignition signal"
            << endl; break;
        case CAN.Button.OnOff.Activity: cout << "Wake up on CAN, Buttons and
            On/Off/Ignition signal" << endl; break;
        case CI.Button.Activity: cout << "Wake up on CI and Button State Change" <<
            endl; break;
        case CI.OnOff.Activity: cout << "Wake up on CI and OnOff Signal State Change"
            << endl; break;
        case CI.Button.OnOff.Activity: cout << "Wake up on CI, Button and OnOff Signal
            State Change" << endl; break;
        case CI.CAN.Button.Activity: cout << "Wake up on CI, CAN and Button State
            Change" << endl; break;
        case CI.CAN.OnOff.Activity: cout << "Wake up on CI, CAN and OnOff Signal State
            Change" << endl; break;
```



```

    case All.Events: cout << "Wake up on all events" << endl; break;
    default: cout << "Error - Undefined StartupTrigger" << endl; break;
  }
}
else
{
  cout << "Error(" << err << ") in function getStartupTriggerConfig: " <<
    GetErrorStringA(err) << endl;
}

```

### Config\_getStartupVoltageConfig()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_get↵
StartupVoltageConfig (
    CONFIGHANDLE ,
    float64_t * voltage )

```

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.

Supported Platform(s): XL, XL5, XM, XM9, V700

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_getSuspendMaxTime()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_get↵
SuspendMaxTime (
    CONFIGHANDLE ,
    uint16_t * maxTime )

```

Get suspend mode maximum time.

Supported Platform(s): XL, XM, XM9 (CC Linux), VC, VA, VS, VI2, V700

#### 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. See the enum eErr for details.

**Config\_getVideoStartupPowerConfig()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Config_getVideoStartupPowerConfig (
    CONFIGHANDLE ,
    uint8_t * config )
```

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.

Supported Platform(s): XL, XL5, XM, XS, XA

**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. See the enum eErr for details.

**Config\_release()**

```
EXTERN_C CCAUXDLLAPI void CCAUXDLL_CALLING_CONV CrossControl::Config_release (
    CONFIGHANDLE )
```

Delete the Config object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Returns**

-

**Example Usage:****Config\_setButtonFunction()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setButtonFunction (
    CONFIGHANDLE ,
    uint8_t button_number,
    ButtonConfigEnum button_config )
```

Set button function configuration

Supported Platform(s): VC, VA, VS, VI2

## Parameters

<i>button_number</i>	Which button to configure (1-MAX_BUTTONS)
<i>button_config</i>	Bitfield for button configuration, see enum ButtonConfigEnum for details.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setCanStartupPowerConfig()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setCanStartupPowerConfig (
    CONFIGHANDLE ,
    CCStatus status )
```

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.

Supported Platform(s): XL, XL5, XM, XS, XA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setDigPowerOutputStartupConfig()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setDigPowerOutputStartupConfig (
    CONFIGHANDLE ,
    PowerOutput output,
    CCStatus enabled )
```

Set power output startup configuration. Power outputs 1 and 2 can be set to be turned on automatically at system startup.

Supported Platform(s): VS

## Parameters

<i>output</i>	Select power output (PowerOutput1 or PowerOutput2).
<i>enabled</i>	Configure the power output to be enabled at startup.

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setExtFanStartupPowerConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set←
ExtFanStartupPowerConfig (
    CONFIGHANDLE ,
    CCStatus status )
```

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.

Supported Platform(s): XL, XL5, XM

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setExtOnOffSigTrigTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set←
ExtOnOffSigTrigTime (
    CONFIGHANDLE ,
    uint32_t triggertime )
```

Set external on/off signal trigger time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**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. See the enum eErr for details.

**Config\_setFrontBtnTrigTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set←
FrontBtnTrigTime (
    CONFIGHANDLE ,
    uint16_t triggertime )
```

Set front button trigger time for long press.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2

## 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.
--------------------	---

NOTE: On VS the triggertime is limited to maximum 14000 ms.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setHeatingTempLimit()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set←
HeatingTempLimit (
    CONFIGHANDLE ,
    int16_t temperature )
```

Set the temperature 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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setLongBeepSettings()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set←
LongBeepSettings (
```

```

CONFIGHANDLE ,
uint16_t duration,
uint16_t frequency,
uint16_t volume )

```

**Set Long Beep Settings** The computer issues a long beep at certain system events, for example a long button press. This beep may be configured or disabled by setting any of the parameters to 0.

Supported Platform(s): VS, V700

Parameters

<i>duration</i>	Long beep duration (ms)
<i>frequency</i>	Long beep frequency (Hz)
<i>volume</i>	Long beep volume (0-2000)

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_setLongButtonPressAction()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_set←
LongButtonPressAction (
    CONFIGHANDLE ,
    PowerAction action )

```

**Set long button press action.** Sets the configured action for a long button press: NoAction, ActionSuspend or ActionShutDown. A long button press is determined by the FrontBtnTrigTime.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Config\_setNextBootMode()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Config_set←
NextBootMode (
    CONFIGHANDLE ,
    BootModeEnum mode )

```

Set BootMode for next boot. Default mode is restored after boot.  
Supported Platform(s): VS, VI2, V700

Parameters

<i>mode</i>	BootMode for next boot.
-------------	-------------------------

Returns

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

### Config.setOnOffSigAction()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setOnOffSigAction (
    CONFIGHANDLE ,
    PowerAction action )
```

Set On/Off signal action. Sets the configured action for an On/Off signal event: NoAction, ActionSuspend or ActionShutDown. An On/Off signal event is determined by the ExtOnOffSigTrigTime.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

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

### Config.setOnOffTriggerMode()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setOnOffTriggerMode (
    CONFIGHANDLE ,
    ConfigOnOffTriggerMode mode )
```

Set OnOff/Ignition/KeySwitch signal trigger mode.

Supported Platform(s): XM9, XL5, XA, XS, VC, VA, VS, VI2, V700

Parameters

<i>mode</i>	Signal trigger mode. See ConfigOnOffTriggerMode for details
-------------	---



**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setOSAliveMonitoring()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setOSAliveMonitoring (
    CONFIGHANDLE ,
    CCStatus enabled )
```

Set OS Alive Monitoring

Supported Platform(s): VS, VI2, V700

**Parameters**

<i>enabled</i>	Enable/Disable OS Alive Monitoring
----------------	------------------------------------

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setPowerOnStartup()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setPowerOnStartup (
    CONFIGHANDLE ,
    CCStatus status )
```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setRS485Enabled()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set←
RS485Enabled (
    CONFIGHANDLE ,
    RS4XXPort port,
    bool enabled )
```

Set RS485 mode enabled or disabled for RS4XX port.

Supported Platform(s): XA, XS

## Parameters

<i>port</i>	RS4XX port (RS4XXPort1-4)
<i>enabled</i>	RS485 enabled (true/false)

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setShortBeepSettings()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set←
ShortBeepSettings (
    CONFIGHANDLE ,
    uint16_t duration,
    uint16_t frequency,
    uint16_t volume )
```

Set Short Beep Settings The computer issues a short beep at certain system events such as boot or button press. This beep may be configured or disabled by setting any of the parameters to 0.

Supported Platform(s): VS, V700

## Parameters

<i>duration</i>	Short beep duration (ms)
<i>frequency</i>	Short beep frequency (Hz)
<i>volume</i>	Short beep volume (0-2000)

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setShortButtonPressAction()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setShortButtonPressAction (
    CONFIGHANDLE ,
    PowerAction action )
```

Set short button press action. Sets the configured action for a short button press: NoAction, ActionSuspend or ActionShutDown.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

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

**Config\_setStartupTriggerConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setStartupTriggerConfig (
    CONFIGHANDLE ,
    TriggerConf conf )
```

Set Start-up trigger configuration. Should the front button and/or the external on/off (ignition) signal be enabled as triggers for startup and wake up from suspended mode? VC,VA platforms: CI state change and Can activity also available as wakeup triggers from suspend mode. V700 platform: Only external on/off (ignition signal) is accepted. See enum TriggerConf for more details.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Parameters**

<i>conf</i>	See enum TriggerConf.
-------------	-----------------------

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setStartupVoltageConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setStartupVoltageConfig (
    CONFIGHANDLE ,
    float64_t voltage )
```

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.

Supported Platform(s): XL, XL5, XM, XM9, V700

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Config\_setSuspendMaxTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setSuspendMaxTime (
    CONFIGHANDLE ,
    uint16_t maxTime )
```

Set suspend mode maximum time.

Supported Platform(s): XL, XM, XM9 (CC Linux), VC, VA, VS, VI2, V700

**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. See the enum eErr for details.

**Config\_setVideoStartupPowerConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_setVideoStartupPowerConfig (
    CONFIGHANDLE ,
    uint8_t config )
```

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.

Supported Platform(s): XL, XM, XS, XA

## 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. See the enum eErr for details.

**GetConfig()**

```
EXTERN_C CCAUXDLL_API CONFIGHANDLE CCAUXDLL_CALLING_CONV CrossControl::GetConfig ( )
```

Factory function that creates instances of the Config object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Returns

CONFIGHANDLE to an allocated Config object. The returned handle needs to be deallocated using the [Config\\_release\(CONFIGHANDLE\)](#) 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);
Config_release(pConfig);
```

**19.10 Diagnostic functions****Functions**

- EXTERN\_C CCAUXDLL\_API DIAGNOSTICHANDLE CCAUXDLL\_CALLING\_CONV [GetDiagnostic](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Diagnostic\\_release](#) (DIAGNOSTICHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Diagnostic\\_getSSTemp](#) (DIAGNOSTICHANDLE, int16\_t \*temperature)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPCBTemp (DIAGNOSTICHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPMTemp (DIAGNOSTICHANDLE, uint8\_t index, int16\_t \*temperature, JidaSensorType \*jst)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getStartupReason (DIAGNOSTICHANDLE, uint16\_t \*reason)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getShutDownReason (DIAGNOSTICHANDLE, uint16\_t \*reason)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getHwErrorStatus (DIAGNOSTICHANDLE, uint16\_t \*errorCode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getTimer (DIAGNOSTICHANDLE, TimerType \*times)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getMinMaxTemp (DIAGNOSTICHANDLE, int16\_t \*minTemp, int16\_t \*maxTemp)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPowerCycles (DIAGNOSTICHANDLE, uint16\_t \*powerCycles)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_clearHwErrorStatus (DIAGNOSTICHANDLE)

### 19.10.1 Detailed Description

Functions in the Diagnostic class

### 19.10.2 Function Documentation

#### Diagnostic\_clearHwErrorStatus()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↵
_clearHwErrorStatus (
    DIAGNOSTICHANDLE )
```

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

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

#### Diagnostic\_getHwErrorStatus()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↵
_getHwErrorStatus (
    DIAGNOSTICHANDLE ,
    uint16_t * errorCode )
```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Diagnostic\_getMinMaxTemp()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↵
_getMinMaxTemp (
    DIAGNOSTICHANDLE ,
    int16_t * minTemp,
    int16_t * maxTemp )
```

Get diagnostic temperature interval of the unit.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

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

### Diagnostic\_getPCBTemp()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↵
_getPCBTemp (
    DIAGNOSTICHANDLE ,
    int16_t * temperature )
```

Get PCB temperature.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Diagnostic\_getPMTemp()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Diagnostic↵
_getPMTemp (
    DIAGNOSTICHANDLE ,
    uint8_t index,
    int16_t * temperature,
    JidaSensorType * jst )
```

Get Processor Module temperature. This temperature is read from the Kontron JIDA API or Congatec CGOS API. These API's also has a number of other functions, please see the JIDA/CGOS 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. CCpilot XM and XL currently has 2 sensors, board and cpu. An error is returned if the index is not supported. CCpilot XM 2.0 supports only one sensor, CPU temperature.
--------------	--

Supported Platform(s): XL, XL5, XM, XM9

## Parameters

<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. See the enum eErr for details.

**Diagnostic\_getPowerCycles()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Diagnostic↵
_getPowerCycles (
    DIAGNOSTICHANDLE ,
    uint16_t * powerCycles )
```



Get number of power cycles.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Diagnostic\_getShutDownReason()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Diagnostic↔
_getShutDownReason (
    DIAGNOSTICHANDLE ,
    uint16_t * reason )
```

Get shutdown reason.

Supported Platform(s): XL, XL5, XM, XM9

Parameters

<i>reason</i>	See <a href="#">DiagnosticCodes.h</a> for shutdown codes.
---------------	---

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Diagnostic\_getSSTemp()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Diagnostic↔
_getSSTemp (
    DIAGNOSTICHANDLE ,
    int16_t * temperature )
```

Get System Supervisor temperature.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

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

**Diagnostic\_getStartupReason()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↵
_getStartupReason (
    DIAGNOSTICHANDLE ,
    uint16_t * reason )
```

Get startup reason.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Parameters**

<i>reason</i>	See <a href="#">DiagnosticCodes.h</a> for startup codes.
---------------	--

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Diagnostic\_getTimer()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↵
_getTimer (
    DIAGNOSTICHANDLE ,
    TimerType * times )
```

Get diagnostic timer.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = Diagnostic.getTimer(pDiagnostic, &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);

```

### Diagnostic\_release()

```

EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↔
_release (

```

```

    DIAGNOSTICHANDLE )

```

Delete the Diagnostic object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Returns

-

Example Usage:

### GetDiagnostic()

```

EXTERN_C CCAUXDLL_API DIAGNOSTICHANDLE CCAUXDLL_CALLING_CONV CrossControl↔
::GetDiagnostic (
    void )

```

Factory function that creates instances of the Diagnostic object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Returns

DIAGNOSTICHANDLE to an allocated Diagnostic object. The returned handle needs to be deallocated using the [Diagnostic\\_release\(DIAGNOSTICHANDLE\)](#) 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);
Diagnostic_release(pDiagnostic);

```

## 19.11 DigIO functions

### Functions

- EXTERN\_C CCAUXDLL\_API DIGIOHANDLE CCAUXDLL\_CALLING\_CONV [GetDigIO](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [DigIO\\_release](#) (DIGIOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [DigIO\\_getDigIO](#) (DIGIOHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [DigIO\\_setDigIO](#) (DIGIOHANDLE, uint8\_t state)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV DigIO\_getDigPowerOutput (DIGIOHANDLE, PowerOutput output, CCStatus \*enabled, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV DigIO\_setDigPowerOutput (DIGIOHANDLE, PowerOutput output, CCStatus enabled)

### 19.11.1 Detailed Description

Functions in the DigIO class

### 19.11.2 Function Documentation

#### DigIO\_getDigIO()

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::DigIO\_getDigIO (

```
    DIGIOHANDLE ,
    uint8_t * status )
```

Get Digital inputs.

Supported Platform(s): XL, XM, XS, XA

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. See the enum eErr for details.

Example Usage:

```
err = DigIO_getDigIO (pDigIO, &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;
}
```

#### DigIO\_getDigPowerOutput()

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::DigIO\_getDigPowerOutput (

```

DIGIOHANDLE ,
PowerOutput output,
CCStatus * enabled,
uint8_t * status )

```

Get digital power output status.

Supported Platform(s): VS

#### Parameters

<i>output</i>	Which output to read
<i>enabled</i>	State of the power output signal.
<i>status</i>	Error status of the power output signal.

Error status: The error status byte can be used to detect certain error conditions. If the error status byte is set to 1, this indicates an error (0 is no error).

If the signal state is Enabled (high), the following error conditions generates an error status: "Short circuit to GND" or "Over temperature". SS monitors the signal and turns it off immediately if an error is indicated when the signal has been set high. This means that "enabled" will be read as disabled and in this case the status may have any value - although in most cases it will be set to no error since the signal is now turned off.

If the signal is disabled (low), these error conditions generates an error status: "↵ Short circuit to VDD" or "Off state open load".

#### Returns

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

#### Example Usage:

#### DigIO\_release()

```

EXTERN_C CCAUXDLLAPI void CCAUXDLLCALLING_CONV CrossControl::DigIO_release
(
    DIGIOHANDLE )

```

Delete the DigIO object.

Supported Platform(s): XL, XM, XS, XA

#### Returns

-

#### Example Usage:

```

DIGIOHANDLE pDigIO = ::GetDigIO();
assert(pDigIO);
list.digital_inputs(pDigIO);
DigIO_release(pDigIO);

```

**DigIO\_setDigIO()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::DigIO_setDigIO (
    DIGIOHANDLE ,
    uint8_t state )
```

Set Digital outputs.

Supported Platform(s): XA, XS

## 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 not used.
--------------	---

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
err = DigIO_setDigIO (pDigIO, inputs);
if (CrossControl::ERR_SUCCESS == err)
{
    cout << "Digital out set to the status read." << endl;
}
else
{
    cout << "Unable to set digital output status." << endl;
}
```

**DigIO\_setDigPowerOutput()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::DigIO_setDigPowerOutput (
    DIGIOHANDLE ,
    PowerOutput output,
    CCStatus enabled )
```

Set Digital power outputs.

Supported Platform(s): VS

## Parameters

<i>output</i>	The output to set
<i>enabled</i>	State of the power output signal.

Outputs 1 and 2 can also be set to be turned on automatically at system startup using: `Config_setDigPowerOutputStartupConfig`.

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:****GetDigIO()**

```
EXTERN_C CCAUXDLL_API DIGIOHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get↔
DigIO (
    void )
```

Factory function that creates instances of the DigIO object.

Supported Platform(s): XL, XM, XS, XA

**Returns**

DIGIOHANDLE to an allocated DigIO object. The returned handle needs to be deallocated using the [DigIO\\_release\(DIGIOHANDLE\)](#) 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);
DigIO_release(pDigIO);
```

## 19.12 FirmwareUpgrade functions

**Functions**

- EXTERN\_C CCAUXDLL\_API FIRMWAREUPGHANDLE CCAUXDLL\_CALLING\_CONV↔  
\_CONV [GetFirmwareUpgrade](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_release](#)  
(FIRMWAREUPGHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startFpgaUpgrade](#)  
(FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startFpgaVerification](#)  
(FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startSSUpgrade](#)  
(FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startSSVerification](#)  
(FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startFrontUpgrade](#)  
(FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startFrontVerification](#)  
(FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startDMCUUpgrade](#)  
(FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking, uint8\_t in-  
dex)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startDMCUVerification (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking, uint8\_t index)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_getUpgradeStatus (FIRMWAREUPGHANDLE, UpgradeStatus \*status, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_shutDown (FIRMWAREUPGHANDLE)

### 19.12.1 Detailed Description

Functions in the FirmwareUpgrade class

### 19.12.2 Function Documentation

#### FirmwareUpgrade\_getUpgradeStatus()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↔
_getUpgradeStatus (
    FIRMWAREUPGHANDLE ,
    UpgradeStatus * status,
    bool blocking )
```

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

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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. See the enum eErr for details.

#### FirmwareUpgrade\_release()

```
EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↔
_release (
    FIRMWAREUPGHANDLE )
```

Delete the FirmwareUpgrade object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700



## Returns

-

## Example Usage:

```
FirmwareUpgrade.release(pFirmwareUpgrade);
```

**FirmwareUpgrade\_shutDown()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↔
_shutDown (
    FIRMWAREUPGHANDLE )
```

Shut down the operating system.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**FirmwareUpgrade\_startDMCUUpgrade()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↔
_startDMCUUpgrade (
    FIRMWAREUPGHANDLE ,
    const char_t * filename,
    bool blocking,
    uint8_t index )
```

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

Supported Platform(s): Yukon

## 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 fpgaUpgradeStatus 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.
<i>index</i>	Device can has more than one Display MCU. This is the index which device to upgrade

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:****FirmwareUpgrade\_startDMCUVerification()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↔
_startDMCUVerification (
    FIRMWAREUPGHANDLE ,
    const char_t * filename,
    bool blocking,
    uint8_t index )
```

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

Supported Platform(s): Yukon

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:****FirmwareUpgrade\_startFpgaUpgrade()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↔
_startFpgaUpgrade (
    FIRMWAREUPGHANDLE ,
    const char_t * filename,
    bool blocking )
```

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.

Supported Platform(s): XL, XM, XS, XA, VS

## 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. See the enum `eErr` for details.

## Example Usage:

```

cout << "Upgrading FPGA" << endl;
for(int i=0;i<max.retries;i++)
{
    // Reinitialize upgrade handle
    FirmwareUpgrade.release(pFirmwareUpgrade);
    pFirmwareUpgrade = GetFirmwareUpgrade();
    assert(pFirmwareUpgrade != NULL);
    err = FirmwareUpgrade.startFpgaUpgrade(pFirmwareUpgrade, path.c_str(), true);
    if (CrossControl::ERR_SUCCESS == err)
    {
        cout << "Upgrade Ok" << endl;
        break;
    }
    else if (CrossControl::ERR_VERIFY_FAILED == err)
    {
        // Reinitialize upgrade handle
        FirmwareUpgrade.release(pFirmwareUpgrade);
        pFirmwareUpgrade = GetFirmwareUpgrade();
        assert(pFirmwareUpgrade != NULL);
        err = FirmwareUpgrade.startFpgaVerification(pFirmwareUpgrade, path.c_str(),
            true);
        if (CrossControl::ERR_SUCCESS == err)
        {
            cout << "Upgrade Ok" << endl;
            break;
        }
    }
    else
    {
        cout << "Error " << err << " in function startFpgaUpgrade: " <<
            GetErrorStringA(err) << std::endl;
    }
}

```

**FirmwareUpgrade\_startFpgaVerification()**

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↔
_startFpgaVerification (
    FIRMWAREUPGHANDLE ,
    const char_t * filename,
    bool blocking )

```

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

Supported Platform(s): XL, XM, XS, XA, VS

## 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. See the enum `eErr` for details.

## Example Usage:

```
cout << "Upgrading FPGA" << endl;
for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    FirmwareUpgrade.release(pFirmwareUpgrade);
    pFirmwareUpgrade = GetFirmwareUpgrade();
    assert(pFirmwareUpgrade != NULL);
    err = FirmwareUpgrade.startFpgaUpgrade(pFirmwareUpgrade, path.c_str(), true);
    if (CrossControl::ERR_SUCCESS == err)
    {
        cout << "Upgrade Ok" << endl;
        break;
    }
    else if (CrossControl::ERR_VERIFY_FAILED == err)
    {
        // Reinitialize upgrade handle
        FirmwareUpgrade.release(pFirmwareUpgrade);
        pFirmwareUpgrade = GetFirmwareUpgrade();
        assert(pFirmwareUpgrade != NULL);
        err = FirmwareUpgrade.startFpgaVerification(pFirmwareUpgrade, path.c_str(),
            true);
        if (CrossControl::ERR_SUCCESS == err)
        {
            cout << "Upgrade Ok" << endl;
            break;
        }
    }
    else
    {
        cout << "Error " << err << " in function startFpgaUpgrade: " <<
            GetErrorStringA(err) << std::endl;
    }
}
```

**FirmwareUpgrade\_startFrontUpgrade()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::FirmwareUpgrade↵
_startFrontUpgrade (
    FIRMWAREUPGHANDLE ,
    const char_t * filename,
    bool blocking )
```

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.

Supported Platform(s): XL, XM, XS, XA

## 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. See the enum `eErr` for details.

## Example Usage:

```

cout << "Upgrading front" << endl;
for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    FirmwareUpgrade_release(pFirmwareUpgrade);
    pFirmwareUpgrade = GetFirmwareUpgrade();
    assert(pFirmwareUpgrade != NULL);
    err = FirmwareUpgrade_startFrontUpgrade(pFirmwareUpgrade, path.c_str(), true);
    if (CrossControl::ERR_SUCCESS == err)
    {
        cout << "Upgrade Ok" << endl;
        break;
    }
    else if (CrossControl::ERR_VERIFY_FAILED == err)
    {
        // Reinitialize upgrade handle
        FirmwareUpgrade_release(pFirmwareUpgrade);
        pFirmwareUpgrade = GetFirmwareUpgrade();
        assert(pFirmwareUpgrade != NULL);
        err = FirmwareUpgrade_startFrontVerification(pFirmwareUpgrade, path.c_str(),
            true);
        if (CrossControl::ERR_SUCCESS == err)
        {
            cout << "Upgrade Ok" << endl;
            break;
        }
    }
    else
    {
        cout << "Error " << err << " in function startFrontUpgrade: " <<
            GetErrorStringA(err) << std::endl;
    }
}

```

**FirmwareUpgrade\_startFrontVerification()**

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↵
_startFrontVerification (
    FIRMWAREUPGHANDLE ,
    const char_t * filename,
    bool blocking )

```

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.

Supported Platform(s): XL, XM, XS, XA

## 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. See the enum `eErr` for details.

## Example Usage:

```

cout << "Upgrading front" << endl;
for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    FirmwareUpgrade_release(pFirmwareUpgrade);
    pFirmwareUpgrade = GetFirmwareUpgrade();
    assert(pFirmwareUpgrade != NULL);
    err = FirmwareUpgrade_startFrontUpgrade(pFirmwareUpgrade, path.c_str(), true);
    if (CrossControl::ERR_SUCCESS == err)
    {
        cout << "Upgrade Ok" << endl;
        break;
    }
    else if (CrossControl::ERR_VERIFY_FAILED == err)
    {
        // Reinitialize upgrade handle
        FirmwareUpgrade_release(pFirmwareUpgrade);
        pFirmwareUpgrade = GetFirmwareUpgrade();
        assert(pFirmwareUpgrade != NULL);
        err = FirmwareUpgrade_startFrontVerification(pFirmwareUpgrade, path.c_str(),
            true);
        if (CrossControl::ERR_SUCCESS == err)
        {
            cout << "Upgrade Ok" << endl;
            break;
        }
    }
    else
    {
        cout << "Error " << err << " in function startFrontUpgrade: " <<
            GetErrorStringA(err) << std::endl;
    }
}

```

**FirmwareUpgrade\_startSSUpgrade()**

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FirmwareUpgrade↔
_startSSUpgrade (
    FIRMWAREUPGHANDLE ,
    const char_t * filename,
    bool blocking )

```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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. See the enum `eErr` for details.

#### Example Usage:

```

cout << "Upgrading SS" << endl;
for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    FirmwareUpgrade.release(pFirmwareUpgrade);
    pFirmwareUpgrade = GetFirmwareUpgrade();
    assert(pFirmwareUpgrade != NULL);
    err = FirmwareUpgrade.startSSUpgrade(pFirmwareUpgrade, path.c_str(), true);
    if (CrossControl::ERR_SUCCESS == err)
    {
        cout << "Upgrade Ok" << endl;
        break;
    }
    else if (CrossControl::ERR_VERIFY_FAILED == err)
    {
        // Reinitialize upgrade handle
        FirmwareUpgrade.release(pFirmwareUpgrade);
        pFirmwareUpgrade = GetFirmwareUpgrade();
        assert(pFirmwareUpgrade != NULL);
        err = FirmwareUpgrade.startSSVerification(pFirmwareUpgrade, path.c_str(),
            true);
        if (CrossControl::ERR_SUCCESS == err)
        {
            cout << "Upgrade Ok" << endl;
            break;
        }
    }
    else
    {
        cout << "Error " << err << " in function startSSUpgrade: " <<
            GetErrorStringA(err) << std::endl;
    }
}

```

#### FirmwareUpgrade.startSSVerification()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::FirmwareUpgrade↵
_startSSVerification (
    FIRMWAREUPGHANDLE ,
    const char_t * filename,
    bool blocking )

```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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. See the enum `eErr` for details.

#### Example Usage:

```

cout << "Upgrading SS" << endl;
for(int i=0;i<max_retries;i++)
{
    // Reinitialize upgrade handle
    FirmwareUpgrade.release(pFirmwareUpgrade);
    pFirmwareUpgrade = GetFirmwareUpgrade();
    assert(pFirmwareUpgrade != NULL);
    err = FirmwareUpgrade.startSSUpgrade(pFirmwareUpgrade, path.c_str(), true);
    if (CrossControl::ERR_SUCCESS == err)
    {
        cout << "Upgrade Ok" << endl;
        break;
    }
    else if (CrossControl::ERR_VERIFY_FAILED == err)
    {
        // Reinitialize upgrade handle
        FirmwareUpgrade.release(pFirmwareUpgrade);
        pFirmwareUpgrade = GetFirmwareUpgrade();
        assert(pFirmwareUpgrade != NULL);
        err = FirmwareUpgrade.startSSVerification(pFirmwareUpgrade, path.c_str(),
            true);
        if (CrossControl::ERR_SUCCESS == err)
        {
            cout << "Upgrade Ok" << endl;
            break;
        }
    }
    else
    {
        cout << "Error " << err << " in function startSSUpgrade: " <<
            GetErrorStringA(err) << std::endl;
    }
}

```

#### GetFirmwareUpgrade()

```

EXTERN_C CCAUXDLL_API FIRMWAREUPGHANDLE CCAUXDLL_CALLING_CONV CrossControl←
::GetFirmwareUpgrade (
    void )

```

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



Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### Returns

FIRMWAREUPGHANDLE to an allocated FirmwareUpgrade object. The returned handle needs to be deallocated using the [FirmwareUpgrade\\_release\(FIRMWAREUPGHANDLE\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

#### Example Usage:

```
FIRMWAREUPGHANDLE pFirmwareUpgrade = GetFirmwareUpgrade();
assert(pFirmwareUpgrade != NULL);
```

## 19.13 FrontLED functions

### Functions

- EXTERN\_C CCAUXDLL\_API [FRONTLEDHANDLE](#) CCAUXDLL\_CALLING\_CONV [GetFrontLED](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [FrontLED\\_release](#) ([FRONTLEDHANDLE](#))
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_getSignal](#) ([FRONTLEDHANDLE](#), [float64\\_t](#) \*frequency, [uint8\\_t](#) \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_getOnTime](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) \*onTime)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_getOffTime](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) \*offTime)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_getIdleTime](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) \*idleTime)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_getNrOfPulses](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) \*nrOfPulses)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_getColor](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) \*red, [uint8\\_t](#) \*green, [uint8\\_t](#) \*blue)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_getStandardColor](#) ([FRONTLEDHANDLE](#), [CCAuxColor](#) \*color)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_getEnabledDuringStartup](#) ([FRONTLEDHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_setSignal](#) ([FRONTLEDHANDLE](#), [float64\\_t](#) frequency, [uint8\\_t](#) dutyCycle)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_setOnTime](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) onTime)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_setOffTime](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) offTime)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_setIdleTime](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) idleTime)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_setNrOfPulses](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) nrOfPulses)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_setColor](#) ([FRONTLEDHANDLE](#), [uint8\\_t](#) red, [uint8\\_t](#) green, [uint8\\_t](#) blue)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [FrontLED\\_setStandardColor](#) ([FRONTLEDHANDLE](#), [CCAuxColor](#) color)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setOff (FRONTLEDHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setEnabledDuringStartup (FRONTLEDHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getBootLEDConfig (FRONTLEDHANDLE, uint8\_t \*red, uint8\_t \*green, uint8\_t \*blue, float32\_t \*frequency, uint8\_t \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setBootLEDConfig (FRONTLEDHANDLE, uint8\_t red, uint8\_t green, uint8\_t blue, float32\_t frequency, uint8\_t dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getPostBootLEDConfig (FRONTLEDHANDLE, uint8\_t \*red, uint8\_t \*green, uint8\_t \*blue, float32\_t \*frequency, uint8\_t \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setPostBootLEDConfig (FRONTLEDHANDLE, uint8\_t red, uint8\_t green, uint8\_t blue, float32\_t frequency, uint8\_t dutyCycle)

### 19.13.1 Detailed Description

Functions in the FrontLED class

### 19.13.2 Function Documentation

#### FrontLED\_getBootLEDConfig()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_get↵
BootLEDConfig (
    FRONTLEDHANDLE ,
    uint8_t * red,
    uint8_t * green,
    uint8_t * blue,
    float32_t * frequency,
    uint8_t * dutyCycle )
```

Get LED configuration during system boot. The behavior of the LED in terms of color (intensity) and blink frequency may be configured during system boot. Note that FrontLED\_setEnabledDuringStartup overrides these settings, it has to be enabled for these settings to apply. Also note that the frequency limitations mentioned in the description of FrontLED\_setSignal also apply here.

Supported Platform(s): VS, VI2, V700

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.
<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. See the enum eErr for details.

**FrontLED\_getColor()**

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::FrontLED\_getColor (

```
    FRONTLEDHANDLE ,
    uint8_t * red,
    uint8_t * green,
    uint8_t * blue )
```

Get front LED color mix.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

On the VC, VA platforms - the blue parameter gets the button backlight intensity (0-15)

**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. See the enum eErr for details.

**Example Usage:**

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

**FrontLED\_getEnabledDuringStartup()**

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::FrontLED\_getEnabledDuringStartup (

```
    FRONTLEDHANDLE ,
    CCStatus * status )
```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**FrontLED\_getIdleTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_get↔
IdleTime (
```

```
    FRONTLEDHANDLE ,
    uint8_t * idleTime )
```

Get front LED idle time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**FrontLED\_getNrOfPulses()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_get↔
NrOfPulses (
```

```
    FRONTLEDHANDLE ,
    uint8_t * nrOfPulses )
```

Get number of pulses during a blink sequence.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**FrontLED\_getOffTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_get↔
OffTime (
```

```
FRONTLEDHANDLE ,
uint8_t * offTime )
```

Get front LED off time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### FrontLED\_getOnTime()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::FrontLED_get↔
OnTime (
```

```
FRONTLEDHANDLE ,
uint8_t * onTime )
```

Get front LED on time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### FrontLED\_getPostBootLEDConfig()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::FrontLED_get↔
PostBootLEDConfig (
```

```
FRONTLEDHANDLE ,
uint8_t * red,
uint8_t * green,
uint8_t * blue,
float32_t * frequency,
uint8_t * dutyCycle )
```

Get LED configuration after system boot. The behavior of the LED in terms of color (intensity) and blink frequency may be configured. The system will set the LED to this state when the operating system has booted. The LED settings may after this

time, at any time, be overridden by other FrontLED functions. Note that FrontLED←  
\_setEnabledDuringStartup overrides these settings, it has to be enabled for these set-  
tings to apply. Also note that the frequency limitations mentioned in the description of  
FrontLED.setSignal also apply here.

Supported Platform(s): VS, VI2, V700

#### 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.
<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. See the enum eErr for  
details.

### FrontLED\_getSignal()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED.get←  
Signal (
    FRONTLEDHANDLE ,
    float64_t * frequency,
    uint8_t * dutyCycle )
```

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

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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. See the enum eErr for  
details.

#### Example Usage:

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

**FrontLED\_getStandardColor()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_get↔
StandardColor (
    FRONTLEDHANDLE ,
    CCAuxColor * color )
```

Get front LED color from a set of standard colors. If the color is not one of the predefined colors, UNDEFINED\_COLOR will be returned. It is not recommended to use this function on the VC or VA platforms.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**FrontLED\_release()**

```
EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::FrontLED.release
(
    FRONTLEDHANDLE )
```

Delete the FrontLED object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Returns

-

## Example Usage:

**FrontLED\_setBootLEDConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED.set↔
BootLEDConfig (
    FRONTLEDHANDLE ,
    uint8_t red,
    uint8_t green,
    uint8_t blue,
    float32_t frequency,
    uint8_t dutyCycle )
```

Set LED configuration during system boot. The behavior of the LED in terms of color (intensity) and blink frequency may be configured during system boot. Note that FrontLED.setEnabledDuringStartup overrides these settings, it has to be enabled for these settings to apply. Also note that the frequency limitations mentioned in the description of FrontLED\_setSignal also apply here.

Supported Platform(s): VS, VI2, V700

## 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.
<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. See the enum eErr for details.

**FrontLED\_setColor()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_setColor (
```

```
    FRONTLEDHANDLE ,
    uint8_t red,
    uint8_t green,
    uint8_t blue )
```

Set front LED color mix.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

On the VC, VA platforms - use the blue parameter to set the button backlight intensity (0-15)

## 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. See the enum eErr for details.

## Example Usage:

```
err = FrontLED_setColor(pFrontLED, red, green, blue);
if (err != ERR.SUCCESS)
{
    cout << "Error(" << err << ") in function setColor: " << GetErrorStringA(err)
        << endl;
}
```

**FrontLED\_setEnabledDuringStartup()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_setEnabledDuringStartup(
```



```
EnabledDuringStartup (
    FRONTLEDHANDLE ,
    CCStatus status )
```

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.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### FrontLED\_setIdleTime()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::FrontLED_setIdleTime (
    FRONTLEDHANDLE ,
    uint8_t idleTime )
```

Get front LED idle time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### FrontLED\_setNrOfPulses()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::FrontLED_setNrOfPulses (
    FRONTLEDHANDLE ,
    uint8_t nrOfPulses )
```

Set front LED number of pulses during a blink sequence.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**FrontLED\_setOff()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_setOff (
```

```
    FRONTLEDHANDLE )
```

Set front LED off.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**FrontLED\_setOffTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_setOffTime (
```

```
    FRONTLEDHANDLE ,
    uint8_t offTime )
```

Set front LED off time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = FrontLED_setOffTime(pFrontLED, 25);
if (err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setOfftime: " <<
        GetErrorStringA(err) << endl;
}
```

**FrontLED\_setOnTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_setOnTime (
```

```
    FRONTLEDHANDLE ,
    uint8_t onTime )
```

Set front LED on time.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

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

### FrontLED\_setPostBootLEDConfig()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_setPostBootLEDConfig (
    FRONTLEDHANDLE ,
    uint8_t red,
    uint8_t green,
    uint8_t blue,
    float32_t frequency,
    uint8_t dutyCycle )
```

Set LED configuration after system boot. The behavior of the LED in terms of color (intensity) and blink frequency may be configured. The system will set the LED to this state when the operating system has booted. The LED settings may after this time, at any time, be overridden by other FrontLED functions. Note that FrontLED\_setEnabledDuringStartup overrides these settings, it has to be enabled for these settings to apply. Also note that the frequency limitations mentioned in the description of FrontLED\_setSignal also apply here.

Supported Platform(s): VS, VI2, V700

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.
<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. See the enum eErr for details.

**FrontLED\_setSignal()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_setSignal (
    FRONTLEDHANDLE ,
    float64_t frequency,
    uint8_t dutyCycle )
```

Set front LED signal.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**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. See the enum eErr for details.

Note: The hardware cannot be set to have an on or off time of the LED that's longer than 2.55s (255\*10ms) Hence there are limitations in this function when using frequencies slower than 0.39Hz. At 0.38Hz, the valid duty cycle range is [3 - 97]. At 0.30Hz, the valid duty cycle range is [24 - 76]. At 0.20Hz, the valid duty cycle range is [49 - 51]. At 0.19Hz and slower, the behavior is undefined for all duty cycles, so this is not allowed to be set. Additionally, the hardware cannot be set to have an on or off time of the LED that's shorter than 10ms. Hence, there are limitations in this function when using high frequencies. At 50 Hz, the valid duty cycle range is [50]. At 30 Hz, the valid duty cycle range is [30-70]. At 10 Hz, the valid duty cycle range is [10-90]. At 2 Hz, the valid duty cycle range is [2-98]. The behavior is undefined outside these ranges but setting 0% or 100% duty cycle will always work, regardless of the frequency. If you need to blink in an unsupported range, it can be done with a software timer instead.

**Example Usage:**

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

**FrontLED\_setStandardColor()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_setStandardColor (
```

```
FRONTLEDHANDLE ,
CCAuxColor color )
```

Set one of the front LED standard colors. It is not recommended to use this function on the VC/VA/VI2 platform.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
err = FrontLED_setStandardColor(pFrontLED, RED);
if (err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function setStandardColor: " <<
        GetErrorStringA(err) << endl;
}
```

### GetFrontLED()

```
EXTERN_C CCAUXDLL_API FRONTLEDHANDLE CCAUXDLL_CALLING_CONV CrossControl::↔
GetFrontLED (
    void )
```

Factory function that creates instances of the FrontLED object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Returns

FRONTLEDHANDLE to an allocated FrontLED object. The returned handle needs to be deallocated using the [FrontLED.release\(FRONTLEDHANDLE\)](#) 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);
FrontLED.release(pFrontLED);
```

## 19.14 Lightsensor functions

### Functions

- EXTERN\_C CCAUXDLL\_API LIGHTSENSORHANDLE CCAUXDLL\_CALLING\_CONV↔  
\_CONV [GetLightsensor](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Lightsensor\\_release](#)  
(LIGHTSENSORHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_getIlluminance](#)  
(LIGHTSENSORHANDLE, uint16\_t \*value)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_getIlluminance2](#) (LIGHTSENSORHANDLE, uint16\_t \*value, uint8\_t \*ch0, uint8\_t \*ch1)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_getAverageIlluminance](#) (LIGHTSENSORHANDLE, uint16\_t \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_startAverageCalc](#) (LIGHTSENSORHANDLE, uint32\_t averageWndSize, uint32\_t rejectWndSize, uint32\_t rejectDeltaInLux, [LightSensorSamplingMode](#) mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_stopAverageCalc](#) (LIGHTSENSORHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_getOperatingRange](#) (LIGHTSENSORHANDLE, [LightSensorOperationRange](#) \*range)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_setOperatingRange](#) (LIGHTSENSORHANDLE, [LightSensorOperationRange](#) range)

### 19.14.1 Detailed Description

Functions in the Lightsensor class

### 19.14.2 Function Documentation

#### GetLightsensor()

```
EXTERN_C CCAUXDLL_API LIGHTSENSORHANDLE CCAUXDLL_CALLING_CONV CrossControl←
::GetLightsensor (
    void )
```

Factory function that creates instances of the Lightsensor object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

Returns

LIGHTSENSORHANDLE to an allocated Lightsensor object. The returned handle needs to be deallocated using the [Lightsensor\\_release\(LIGHTSENSORHANDLE\)](#) 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);
Lightsensor.release (pLightSensor);
```

#### Lightsensor\_getAverageIlluminance()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Lightsensor←
_getAverageIlluminance (
    LIGHTSENSORHANDLE ,
    uint16_t * value )
```

Get average illuminance (light) value from light sensor.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

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

**Lightsensor\_getIlluminance()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Lightsensor↔
_getIlluminance (
    LIGHTSENSORHANDLE ,
    uint16_t * value )
```

Get illuminance (light) value from light sensor.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

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

**Lightsensor\_getIlluminance2()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Lightsensor↔
_getIlluminance2 (
    LIGHTSENSORHANDLE ,
    uint16_t * value,
    uint8_t * ch0,
    uint8_t * ch1 )
```

Get illuminance (light) value from light sensor. The parameters `cho` and `chl` are raw ADC values read from a TAOS TSL2550 lightsensor.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

Parameters

<i>value</i>	Illuminance value (Lux).
<i>cho</i>	Channel0 value. (Not applicable on VC platform - always 0)
<i>chl</i>	Channel1 value. (Not applicable on VC platform - always 0)

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum `eErr` for details.

### Lightsensor\_getOperatingRange()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Lightsensor↔
_getOperatingRange (
    LIGHTSENSORHANDLE ,
    LightSensorOperationRange * range )
```

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. On the VC platform, the ranges correspond to 1000 and 4000 lux maximum value.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code.

### Lightsensor\_release()

```
EXTERN_C CCAUXDLLAPI void CCAUXDLL_CALLING_CONV CrossControl::Lightsensor↔
_release (
    LIGHTSENSORHANDLE )
```

Delete the Lightsensor object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700



## Returns

-

## Example Usage:

```

LIGHTSENSORHANDLE pLightSensor = ::GetLightSensor();
assert(pLightSensor);
ls_example(pLightSensor);
LightSensor_release(pLightSensor);

```

**LightSensor\_setOperatingRange()**

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::LightSensor←
_setOperatingRange (
    LIGHTSENSORHANDLE ,
    LightSensorOperationRange range )

```

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. On the VC platform, the ranges correspond to 1000 and 4000 lux maximum value.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

## Parameters

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

## Returns

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

**LightSensor\_startAverageCalc()**

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::LightSensor←
_startAverageCalc (
    LIGHTSENSORHANDLE ,
    uint32_t averageWndSize,
    uint32_t rejectWndSize,
    uint32_t rejectDeltaInLux,
    LightSensorSamplingMode mode )

```

Start average calculation. The average calculation works by calculating the average from a number of consecutive samples, the average window size. The reject window is used to discard sudden changes or single extreme values of the measurement. If the difference of the maximum value and the minimum value in the number of samples in the reject delta window is larger than the reject delta, those samples are discarded.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

## Parameters

<i>averageWndSize</i>	The average window size in nr of samples.
-----------------------	---

## Parameters

<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. See the enum eErr for details.

## Example Usage:

```
// Start the average calculation background function
// This cannot be used if the automatic backlight function is running.
err = Lightsensor_startAverageCalc(pLightSensor, 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;
}
```

**Lightsensor\_stopAverageCalc()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Lightsensor↔
_stopAverageCalc (
    LIGHTSENSORHANDLE )
```

Stop average calculation.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, V700

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

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

**19.15 Power functions****Functions**

- EXTERN\_C CCAUXDLL\_API POWERHANDLE CCAUXDLL\_CALLING\_CONV↔  
GetPower (void)

- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Power\\_release](#) ([POWERHANDLE](#))
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getBLPowerStatus](#) ([POWERHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getCanPowerStatus](#) ([POWERHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getVideoPowerStatus](#) ([POWERHANDLE](#), [uint8\\_t](#) \*videoStatus)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getExtFanPowerStatus](#) ([POWERHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getButtonPowerTransitionStatus](#) ([POWERHANDLE](#), [ButtonPowerTransitionStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getVideoOCDStatus](#) ([POWERHANDLE](#), [OCDStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getCanOCDStatus](#) ([POWERHANDLE](#), [OCDStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_setBLPowerStatus](#) ([POWERHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_setCanPowerStatus](#) ([POWERHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_setVideoPowerStatus](#) ([POWERHANDLE](#), [uint8\\_t](#) status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_setExtFanPowerStatus](#) ([POWERHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_ackPowerRequest](#) ([POWERHANDLE](#))

### 19.15.1 Detailed Description

Functions in the Power class

### 19.15.2 Function Documentation

#### GetPower()

```
EXTERN_C CCAUXDLL_API POWERHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get↔
Power (
    void )
```

Factory function that creates instances of the Power object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### Returns

[POWERHANDLE](#) to an allocated Power object. The returned handle needs to be deallocated using the [Power\\_release\(POWERHANDLE\)](#) 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);
Power_release(pPower);
```

**Power\_ackPowerRequest()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Power_ackPowerRequest (
    POWERHANDLE )
```

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 [CrossControl](#) 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`

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

**Returns**

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

**Power\_getBLPowerStatus()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Power_getBLPowerStatus (
    POWERHANDLE ,
    CCStatus * status )
```

Get backlight power status.

Supported Platform(s): XL, XL5, XM, XM9

**Parameters**

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

**Returns**

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

**Example Usage:**

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

**Power\_getButtonPowerTransitionStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_getButtonPowerTransitionStatus (
    POWERHANDLE ,
    ButtonPowerTransitionStatus * status )
```

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

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Power\_getCanOCDDStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_getCanOCDDStatus (
    POWERHANDLE ,
    OCDDStatus * status )
```

Get Can power overcurrent detection status. Find out if the Can power supervision has detected overcurrent, likely caused by short circuit problems. The overcurrent detection system will immediately turn of the power if such a condition occurs. If the overcurrent remains, Can power is turned off permanently until the unit is restarted. Up to 5 consecutive over-current conditions needed until power is turned off completely. If application software turns off and on the power, the failure counter will be reset.

Supported Platform(s): XL, XL5, XM, XS, XA

## Parameters

<i>status</i>	The current overcurrent detection status
---------------	--

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
cout << "Checking overcurrent status... " << endl;
OCDDStatus ocdstatus;
err = Power_getCanOCDDStatus(pPower, &ocdstatus);
if (err == ERR_NOT_SUPPORTED)
{
    cout << "Not supported." << endl;
}
```

```

else if (err != ERR.SUCCESS)
{
    cout << "Error(" << err << ") in function Power_getCanOCDStatus: " <<
        GetErrorStringA(err) << endl;
}
else
{
    cout << "Power_getCanOCDStatus: Can OCD status is: ";
    switch(oCdstatus)
    {
        case OCD_OK: cout << "OCD_OK" << std::endl; break;
        case OCD_OC: cout << "OCD_OC" << std::endl; break;
        case OCD_POWER_OFF: cout << "OCD_POWER_OFF" << std::endl; break;
        default: cout << "ERROR" << std::endl; break;
    }
}
}

```

### Power\_getCanPowerStatus()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Power_getCan↔
PowerStatus (
    POWERHANDLE ,
    CCStatus * status )

```

Get can power status.

Supported Platform(s): XL, XL5, XM, XS, XA

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Power\_getExtFanPowerStatus()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Power_getExt↔
FanPowerStatus (
    POWERHANDLE ,
    CCStatus * status )

```

Get external fan power status.

Supported Platform(s): XL, XL5, XM

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Power\_getVideoOCDStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_getVideoOCDStatus (
    POWERHANDLE ,
    OCDStatus * status )
```

Get Video power overcurrent detection status. Find out if the video power supervision has detected overcurrent, likely caused by short circuit problems. The overcurrent detection system will immediately turn of the power if such a condition occurs. If the overcurrent remains, video power is turned off permanently until the unit is restarted. Up to 5 consecutive over-current conditions needed until power is turned off completely. If application software turns off and on the power, the failure counter will be reset.

Supported Platform(s): XL, XM, XS, XA

## Parameters

<i>status</i>	The current overcurrent detection status
---------------	--

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
err = Power_getVideoOCDStatus(pPower, &ocdstatus);
if (err == ERR_NOT_SUPPORTED)
{
    /* Don't print anything */
}
else
if (err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function Power_getVideoOCDStatus: " <<
        GetErrorStringA(err) << endl;
}
else
{
    cout << "Power_getVideoOCDStatus: Video OCD status is: ";
    switch(ocdstatus)
    {
        case OCD_OK: cout << "OCD_OK" << std::endl; break;
        case OCD_OC: cout << "OCD_OC" << std::endl; break;
        case OCD_POWER_OFF: cout << "OCD_POWER_OFF" << std::endl; break;
        default: cout << "ERROR" << std::endl; break;
    }
}
```

**Power\_getVideoPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_getVideoPowerStatus (
    POWERHANDLE ,
    uint8_t * videoStatus )
```

Get Video power status.

Supported Platform(s): XL, XL5, XM, XS, XA

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. See the enum eErr for details.

Example Usage:

```
err = Power_getVideoPowerStatus(pPower, &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 Power_getVideoPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

**Power\_release()**

```
EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::Power_release
(
    POWERHANDLE )
```

Delete the Power object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Returns

-

Example Usage:

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

**Power\_setBLPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_setBLPower←
Status (
    POWERHANDLE ,
    CCStatus status )
```

Set backlight power status.

Supported Platform(s): XL, XL5, XM, XM9



## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## 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 = Power_setBLPowerStatus(pPower, Disabled);
cin.sync();
cin.get();
err = Power_setBLPowerStatus(pPower, Enabled);
if (err != ERR_SUCCESS)
{
    cout << "Error(" << err << ") in function Power_setBLPowerStatus: " <<
        GetErrorStringA(err) << endl;
}
```

**Power\_setCanPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_setCanPowerStatus (
    POWERHANDLE ,
    CCStatus status )
```

Set can power status.

Supported Platform(s): XL, XL5, XM, XS, XA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Power\_setExtFanPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_setExtFanPowerStatus (
    POWERHANDLE ,
    CCStatus status )
```

Set external fan power status.

Supported Platform(s): XL, XL5, XM

## Parameters

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

## Returns

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

**Power\_setVideoPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_setVideo↵
PowerStatus (
    POWERHANDLE ,
    uint8_t status )
```

Set Video power status.

Supported Platform(s): XL, XM, XS, XA

## 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. See the enum eErr for details.

**19.16 PowerMgr functions****Functions**

- EXTERN\_C CCAUXDLL\_API POWERMGRHANDLE CCAUXDLL\_CALLING\_CONV GetPowerMgr (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV PowerMgr\_release (POWERMGRHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_registerControlledSuspendOrShutDown (POWERMGRHANDLE, PowerMgrConf conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_getConfiguration (POWERMGRHANDLE, PowerMgrConf \*conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_getPowerMgrStatus (POWERMGRHANDLE, PowerMgrStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_setAppReadyForSuspendOrShutDown (POWERMGRHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_hasResumed (POWERMGRHANDLE, bool \*resumed)

## 19.16.1 Detailed Description

Functions in the PowerMgr class

## 19.16.2 Function Documentation

### GetPowerMgr()

```
EXTERN_C CCAUXDLLAPI POWERMGRHANDLE CCAUXDLLCALLING_CONV CrossControl::GetPowerMgr (
    void )
```

Factory function that creates instances of the PowerMgr object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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;
Battery_isBatteryPresent(pBattery, &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;
Battery_release(pBattery);
// 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 = PowerMgr_registerControlledSuspendOrShutDown(pPowerMgr, (PowerMgrConf)
    suspendConfiguration);
cout << "suspendConfiguration " << suspendConfiguration << endl;
if (err == ERR_SUCCESS)
    cout << "Registered to powerMgr." << endl;
else
    cout << "Error(" << err << ") in function registerControlledSuspendOrShutDown:
        " << GetErrorStringA(err) << endl;
test_powermgr(pPowerMgr);
PowerMgr_release(pPowerMgr);
```

### PowerMgr\_getConfiguration()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::PowerMgr_getConfiguration (
```

```
POWERMGRHANDLE ,
PowerMgrConf * conf )
```

Get the configuration that is in use.

Supported Platform(s): XL, XL5 ,XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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 = PowerMgr_getConfiguration(pPowerMgr, &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;
}
```

### PowerMgr\_getPowerMgrStatus()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::PowerMgr_get↔
PowerMgrStatus (
```

```
POWERMGRHANDLE ,
PowerMgrStatus * status )
```

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

Supported Platform(s): XL, XL5 ,XM, XM9, XS, XA, VC, VA, VS, VI2, V700

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 = PowerMgr.getPowerMgrStatus(pPowerMgr, &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 = PowerMgr.setAppReadyForSuspendOrShutdown(pPowerMgr);
                    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 = PowerMgr.setAppReadyForSuspendOrShutdown(pPowerMgr);
                    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 = PowerMgr.hasResumed(pPowerMgr, &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 = PowerMgr_getConfiguration(pPowerMgr, &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
PowerMgr_registerControlledSuspendOrShutdown(pPowerMgr, 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;
}
}
}

```

### PowerMgr\_hasResumed()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLL-CALLING-CONV CrossControl::PowerMgr_hasResu↵
Resumed (
    POWERMGRHANDLE ,
    bool * resumed )

```

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`.

Supported Platform(s): XL, XL5, XM, XM9 (CC Linux), VC, VA, VS, VI2, V700

### Returns

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

### Example Usage:

```

while(1)
{
    OSSleep(500);
    PowerMgrStatus status;
    err = PowerMgr_getPowerMgrStatus(pPowerMgr, &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 = PowerMgr.setAppReadyForSuspendOrShutdown(pPowerMgr);
        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 = PowerMgr.setAppReadyForSuspendOrShutdown(pPowerMgr);
        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 = PowerMgr.hasResumed(pPowerMgr, &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 = PowerMgr.getConfiguration(pPowerMgr, &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
        PowerMgr.registerControlledSuspendOrShutDown(pPowerMgr, 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;
}
}
}

```

### PowerMgr\_registerControlledSuspendOrShutDown()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PowerMgr_register↵
ControlledSuspendOrShutDown (
    POWERMGRHANDLE ,
    PowerMgrConf conf )

```

Configure the PowerMgr. Call this function once initially to turn on the functionality.

Supported Platform(s): XL, XL5 ,XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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;
Battery_isBatteryPresent(pBattery, &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;
Battery_release(pBattery);
// 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 = PowerMgr.registerControlledSuspendOrShutdown(pPowerMgr, (PowerMgrConf)
suspendConfiguration);
cout << "suspendConfiguration " << suspendConfiguration << endl;
if (err == ERR_SUCCESS)
    cout << "Registered to powerMgr." << endl;
else
    cout << "Error(" << err << ") in function registerControlledSuspendOrShutdown:
    " << GetErrorStringA(err) << endl;
test_powermgr(pPowerMgr);
PowerMgr.release(pPowerMgr);

```

### PowerMgr.release()

```

EXTERN_C CCAUXDLLAPI void CCAUXDLLCALLING_CONV CrossControl::PowerMgr.release
(
    POWERMGRHANDLE )

```

Delete the PowerMgr object.

Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### 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;
Battery_IsBatteryPresent(pBattery, &bBatt);
if (bBatt) // Ask user wicch 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;
Battery.release(pBattery);
// 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 = PowerMgr.registerControlledSuspendOrShutdown(pPowerMgr, (PowerMgrConf)
suspendConfiguration);
cout << "suspendConfiguration " << suspendConfiguration << endl;
if (err == ERR_SUCCESS)
    cout << "Registered to powerMgr." << endl;
else
    cout << "Error(" << err << ") in function registerControlledSuspendOrShutdown:
    " << GetErrorStringA(err) << endl;
test_powermgr(pPowerMgr);
PowerMgr.release(pPowerMgr);

```

### PowerMgr.setAppReadyForSuspendOrShutdown()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::PowerMgr.set↔
AppReadyForSuspendOrShutdown (

```

`POWERMGRHANDLE` )

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.

Supported Platform(s): XL, XL5 ,XM, XM9, XS, XA, VC, VA, VS, VI2, V700

#### Returns

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

#### Example Usage:

```
while(1)
{
    OSSleep(500);
    PowerMgrStatus status;
    err = PowerMgr_getPowerMgrStatus(pPowerMgr, &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 = PowerMgr_setAppReadyForSuspendOrShutdown(pPowerMgr);
                    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 = PowerMgr_setAppReadyForSuspendOrShutdown(pPowerMgr);
                    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 = PowerMgr.hasResumed(pPowerMgr, &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 = PowerMgr.getConfiguration(pPowerMgr, &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
        PowerMgr.registerControlledSuspendOrShutDown(pPowerMgr, 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;
}
}
}

```

## 19.17 PWMOut functions

### Functions

- EXTERN\_C CCAUXDLL\_API PWMOUTHANDLE CCAUXDLL\_CALLING↔  
\_CONV GetPWMOut (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV PWMOut\_release  
(PWMOUTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_setPWMOutputChannelDutyC  
(PWMOUTHANDLE, uint8\_t channel, uint8\_t duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_setPWMOutputChannelFreque  
(PWMOUTHANDLE, uint8\_t channel, float32\_t frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_getPWMOutputChannelDutyC  
(PWMOUTHANDLE, uint8\_t channel, uint8\_t \*duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_getPWMOutputChannelFreque  
(PWMOUTHANDLE, uint8\_t channel, float32\_t \*frequency)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_getPWMOutputStatus (PWMOUTHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_setPWMOutOff (PWMOUTHANDLE, uint8\_t channel)

### 19.17.1 Detailed Description

Functions in the PWMOut class

### 19.17.2 Function Documentation

#### GetPWMOut()

```
EXTERN_C CCAUXDLL_API PWMOUTHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get←
PWMOut (
    void )
```

Factory function that creates instances of the PWMOut object.

Supported Platform(s): VC, VA, VI2

#### Returns

PWMOUTHANDLE to an allocated PWMOut object. The returned handle needs to be deallocated using the PWMOut\_release(PWMOUTHANDLE) method when it's no longer needed. Returns NULL if it fails to allocate memory.

#### Example Usage:

```
PWMOUTHANDLE pPwmOut = ::GetPWMOut();
assert(pPwmOut);
pwmout_example(pPwmOut);
PWMOut_release(pPwmOut);
```

#### PWMOut\_getPWMOutputChannelDutyCycle()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PWMOut_get←
PWMOutputChannelDutyCycle (
    PWMOUTHANDLE ,
    uint8_t channel,
    uint8_t * duty_cycle )
```

Get PWM Output channel duty cycle

Supported Platform(s): VC, VA, VI2

#### Parameters

<i>channel</i>	Which channel to get value from There are two output channels, 1 or 2.
<i>duty_cycle</i>	The read back duty cycle value NOTE: For low side PWM outputs, a duty cycle of 60% means 60% low, 40% high For high side PWM outputs, a duty cycle of 60% means 60% high, 40% low

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
unsigned char duty;
err = PWMOut.getPWMOutputChannelDutyCycle(pPwmOut, 1, &duty);
if (err != ERR_SUCCESS)
{
    cout << "PWMOut.getPWMOutputChannelDutyCycle: " << GetErrorStringA(err) <<
        std::endl;
}
else
{
    cout << "PWMOut.getPWMOutputChannelDutyCycle channel 1: " << (int)duty << "%
        duty cycle" << std::endl;
}
```

**PWMOut.getPWMOutputChannelFrequency()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PWMOut.get←
PWMOutputChannelFrequency (
    PWMOUTHANDLE ,
    uint8_t channel,
    float32_t * frequency )
```

Get PWM Output frequency for a channel

Supported Platform(s): VC, VA, VI2

**Parameters**

<i>channel</i>	Which channel to set There are two output channels, 1 or 2.
<i>frequency</i>	0.0 - 5000.0 Hz frequency value

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
float frequency;
err = PWMOut.getPWMOutputChannelFrequency(pPwmOut, 1, &frequency);
if (err != ERR_SUCCESS)
{
    cout << "PWMOut.getPWMOutputChannelFrequency: " << GetErrorStringA(err) <<
        std::endl;
}
else
{
    cout << "PWMOut.getPWMOutputChannelFrequency channel 1: " << std::fixed <<
        frequency << "Hz" << std::endl;
}
```

**PWMOut.getPWMOutputStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PWMOut.get←
PWMOutputStatus (
```

```

    PWMOUTHANDLE ,
    uint8_t * status )

```

Get PWM Output status

Supported Platform(s): VC, VA, VI2

#### Parameters

<i>status</i>	Read back status value Bit 0 represents PWM Output channel 1. Bit 1 represents PWM Output channel 2. If bit is set, it means unconnected, short to ground or over temperature detected. The output will be turned off when the error occurs. The error status remains until the output is turned on successfully.
---------------	---

#### Returns

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

#### Example Usage:

```

unsigned char status;
err = PWMOut_getPWMOutputStatus(pPwmOut, &status);
if (err != ERR.SUCCESS)
{
    cout << "PWMOut_getPWMOutputStatus: " << GetErrorStringA(err) << std::endl;
}
else
{
    if (status & 0x01)
        cout << "PWMOut_getPWMOutputStatus: Status Not OK for channel 1" <<
            std::endl;
    if (status & 0x02)
        cout << "PWMOut_getPWMOutputStatus: Status Not OK for channel 2" <<
            std::endl;
    if ((status & 0x03) == 0)
        cout << "PWMOut_getPWMOutputStatus: Status OK for both channels" <<
            std::endl;
}

```

#### PWMOut\_release()

```

EXTERN_C CCAUXDLLAPI void CCAUXDLLCALLING_CONV CrossControl::PWMOut_release
(

```

```

    PWMOUTHANDLE )

```

Delete the PWMOut object.

Supported Platform(s): VC, VA, VI2

#### Returns

-

#### Example Usage:

```

PWMOUTHANDLE pPwmOut = ::GetPWMOut();
assert(pPwmOut);
pwmout_example(pPwmOut);
PWMOut_release(pPwmOut);

```

**PWMOut\_setPWMOutOff()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PWMOut_setPWMOutOff (
    PWMOUTHANDLE ,
    uint8_t channel )
```

Turn off a PWM Output channel. This function sets both frequency and duty cycle to 0.

Supported Platform(s): VC, VA, VI2

Parameters

<i>channel</i>	Which channel to set
----------------	----------------------

Returns

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

Example Usage:

```
err = PWMOut_setPWMOutOff(pPwmOut, 1);
if (err != ERR.SUCCESS)
{
    cout << "PWMOut_setPWMOutOff: " << GetErrorStringA(err) << std::endl;
}
else
{
    cout << "PWMOut_setPWMOutOff channel 1 turned off" << std::endl;
}
```

**PWMOut\_setPWMOutputChannelDutyCycle()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PWMOut_setPWMOutputChannelDutyCycle (
    PWMOUTHANDLE ,
    uint8_t channel,
    uint8_t duty_cycle )
```

Set PWM Output Duty cycle for a channel

Supported Platform(s): VC, VA, VI2

Parameters

<i>channel</i>	Which channel to set There are two output channels, 1 or 2.
<i>duty_cycle</i>	Which duty cycle (0-100 %) to use NOTE: For low side PWM outputs, a duty cycle of 60% means 60% low, 40% high For high side PWM outputs, a duty cycle of 60% means 60% high, 40% low

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = PWMOut.setPWMOutputChannelDutyCycle(pPwmOut, 1, 50);
if (err != ERR_SUCCESS)
{
    cout << "setPWMOutputChannelDutyCycle: " << GetErrorStringA(err) << std::endl;
}
else
{
    cout << "setPWMOutputChannelDutyCycle: channel 1 set to 50% duty cycle" <<
        std::endl;
}
```

**PWMOut.setPWMOutputChannelFrequency()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PWMOut.set←
PWMOutputChannelFrequency (
    PWMOUTHANDLE ,
    uint8_t channel,
    float32_t frequency )
```

Set PWM Output frequency for a channel

Supported Platform(s): VC, VA, VI2

**Parameters**

<i>channel</i>	Which channel to set There are two output channels, 1 or 2.
<i>frequency</i>	0.0 - 5000.0 Hz frequency value

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = PWMOut.setPWMOutputChannelFrequency(pPwmOut, 1, (float)100.0);
if (err != ERR_SUCCESS)
{
    cout << "PWMOut.setPWMOutputChannelFrequency: " << GetErrorStringA(err) <<
        std::endl;
}
else
{
    cout << "PWMOut.setPWMOutputChannelFrequency: channel 1 set to 100Hz" <<
        std::endl;
}
```

**19.18 Smart functions****Functions**

- EXTERN\_C CCAUXDLL\_API SMARTHANDLE CCAUXDLL\_CALLING\_CONV GetSmart (void)



- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Smart\_release (SMARTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getRemainingLifeTime (SMARTHANDLE, uint8\_t \*lifetimepercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getRemainingLifeTime2 (SMARTHANDLE, uint8\_t \*lifetimepercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getDeviceSerial (SMARTHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getDeviceSerial2 (SMARTHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getInitialTime (SMARTHANDLE, time\_t \*time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getInitialTime2 (SMARTHANDLE, time\_t \*time)

### 19.18.1 Detailed Description

Functions in the Smart class

### 19.18.2 Function Documentation

#### GetSmart()

```
EXTERN_C CCAUXDLL_API SMARTHANDLE CCAUXDLL_CALLING_CONV CrossControl::GetSmart (
    void )
```

Factory function that creates instances of the Smart object.

Supported Platform(s): XL, XM, XM9

#### 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);
Smart_release(pSmart);
```

#### Smart\_getDeviceSerial()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Smart_getDeviceSerial (
    SMARTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get serial number of the secondary storage device.

Supported Platform(s): XL, XM, XM9

## 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. See the enum eErr for details.

## Example Usage:

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

**Smart\_getDeviceSerial2()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Smart_getDeviceSerial2 (
    SMARTHANDLE ,
    char_t * buff,
    int32_t len )
```

Get serial number of the second secondary storage device. Use this function to access the second card if the the device uses two cards.

Supported Platform(s): XL

## 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. ERR\_CODE\_NOT\_EXIST if only one card is available on XL platform. See the enum eErr for details.

## Example Usage:

```
char serial[21];
err = Smart_getDeviceSerial2 (pSmart, serial, sizeof(serial));
```

```

if (ERR.SUCCESS == err)
{
    cout << "Device serial number: " << serial << endl;
}
else if (ERR.NOT_SUPPORTED == err)
{
    cout << "Smart_getDeviceSerial2 is not supported on this platform" << endl;
}
else
{
    cout << "Error(" << err << ") in function getDeviceSerial: " <<
        GetErrorStringA(err) << endl;
}
}

```

### Smart\_getInitialTime()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Smart_getInitialTime (
    SMARTHANDLE ,
    time_t * time )

```

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.

Supported Platform(s): XL, XM, XM9

#### 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. See the enum eErr for details.

#### Example Usage:

```

time_t initialTime;
struct tm * timeinfo;
err = Smart_getInitialTime (pSmart, &initialTime);
if (ERR.SUCCESS == err)
{
    cout << "Device was initially timestamped on: ";
    timeinfo = localtime (&initialTime);
    if (timeinfo)
    {
        cout << asctime(timeinfo) << endl;
    }
    else
    {
        cout << "Error getting initial time" << endl;
    }
}
else
{
    cout << "Error(" << err << ") in function getInitialTime: " <<
        GetErrorStringA(err) << endl;
}
}

```

**Smart\_getInitialTime2()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Smart_getInitialTime2 (
    SMARTHANDLE ,
    time_t * time )
```

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.

Use this function to access the second card if the the device uses two cards.

Supported Platform(s): XL

## 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. ERR\_CODE\_NOT\_EXIST if only one card is available on XL platform. See the enum eErr for details.

## Example Usage:

```
time_t initialTime;
struct tm * timeinfo;
err = Smart_getInitialTime2 (pSmart, &initialTime);
if (ERR_SUCCESS == err)
{
    cout << "Device was initially timestamped on: ";
    timeinfo = localtime (&initialTime);
    if(timeinfo)
    {
        cout << asctime(timeinfo) << endl;
    }
    else
    {
        cout << "Error getting initial time" << endl;
    }
}
else if (ERR_NOT_SUPPORTED == err)
{
    cout << "Smart_getInitialTime2 is not supported on this platform" << endl;
}
else
{
    cout << "Error(" << err << ") in function getInitialTime: " <<
        GetErrorStringA(err) << endl;
}
```

**Smart\_getRemainingLifeTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Smart_getRemainingLifeTime (
    SMARTHANDLE ,
    uint8_t * lifetimepercent )
```

Get remaining lifetime of the secondary storage device.  
Supported Platform(s): XL, XM, XM9

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

#### Example Usage:

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

#### Smart\_getRemainingLifeTime2()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Smart_getRemaining←
LifeTime2 (
    SMARTHANDLE ,
    uint8_t * lifetimepercent )
```

Get remaining lifetime of the second secondary storage device. Use this function to access the second card if the the device uses two cards.

Supported Platform(s): XL

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. ERR\_CODE\_NOT←  
\_EXIST if only one card is available on XL platform. See the enum eErr for details.

#### Example Usage:

```
unsigned char life;
err = Smart_getRemainingLifeTime2 (pSmart, &life);
if (ERR_SUCCESS == err)
{
    cout << "Estimated remaining lifetime: " << (int)life << "%" << endl;
}
else if (ERR_NOT_SUPPORTED == err)
```

```

{
    cout << "Smart_getRemainingLifeTime2 is not supported on this platform" <<
        endl;
}
else
{
    cout << "Error(" << err << ") in function getRemainingLifeTime: " <<
        GetErrorStringA(err) << endl;
}

```

### Smart\_release()

```

EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::Smart_release
(
    SMARTHANDLE
)

```

Delete the Smart object.

Supported Platform(s): XL, XM, XM9

### Returns

-

### Example Usage:

```

SMARTHANDLE pSmart = ::GetSmart();
assert(pSmart);
show_card_data(pSmart);
Smart_release(pSmart);

```

## 19.19 SoftKey functions

### Functions

- EXTERN\_C CCAUXDLL\_API [SOFTKEYHANDLE](#) CCAUXDLL\_CALLING\_CONV ↔  
\_CONV [GetSoftKey](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [SoftKey\\_release](#)  
([SOFTKEYHANDLE](#))
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [SoftKey\\_getStatus](#)  
([SOFTKEYHANDLE](#), [uint16\\_t](#) \*value)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightIntensity](#)  
([SOFTKEYHANDLE](#), [uint8\\_t](#) key, [uint8\\_t](#) intensity)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightIntensity](#)  
([SOFTKEYHANDLE](#), [uint8\\_t](#) key, [uint8\\_t](#) \*intensity)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [SoftKey\\_setMultipleBacklightIntensities](#)  
([SOFTKEYHANDLE](#), [uint8\\_t](#) \*keys, [uint8\\_t](#) \*intensities, [uint8\\_t](#) array\_size)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [SoftKey\\_getMultipleBacklightIntensities](#)  
([SOFTKEYHANDLE](#), [uint8\\_t](#) \*keys, [uint8\\_t](#) \*intensities, [uint8\\_t](#) array\_size)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightSignal](#)  
([SOFTKEYHANDLE](#), [float64\\_t](#) \*frequency, [uint8\\_t](#) \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightOnTime](#)  
([SOFTKEYHANDLE](#), [uint8\\_t](#) \*onTime)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightOffTime](#)  
([SOFTKEYHANDLE](#), [uint8\\_t](#) \*offTime)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightIdleTime (SOFTKEYHANDLE, uint8\_t \*idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightNrOfPulses (SOFTKEYHANDLE, uint8\_t \*nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightSignal (SOFTKEYHANDLE, float64\_t frequency, uint8\_t dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightOnTime (SOFTKEYHANDLE, uint8\_t onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightOffTime (SOFTKEYHANDLE, uint8\_t offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightIdleTime (SOFTKEYHANDLE, uint8\_t idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightNrOfPulses (SOFTKEYHANDLE, uint8\_t nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightOff (SOFTKEYHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightEnabledDuringStart (SOFTKEYHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightEnabledDuringStart (SOFTKEYHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBootBacklightConfig (SOFTKEYHANDLE, uint8\_t \*bootIntensity, float32\_t \*bootFrequency, uint8\_t \*bootDutyCycle, uint8\_t \*postBootIntensity, float32\_t \*postBootFrequency, uint8\_t \*postBootDutyCycle, CCStatus \*postBootConfig)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBootBacklightConfig (SOFTKEYHANDLE, uint8\_t bootIntensity, float32\_t bootFrequency, uint8\_t bootDutyCycle, uint8\_t postBootIntensity, float32\_t postBootFrequency, uint8\_t postBootDutyCycle, CCStatus postBootConfig)

### 19.19.1 Detailed Description

Functions in the SoftKey class

### 19.19.2 Function Documentation

#### GetSoftKey()

```
EXTERN_C CCAUXDLL_API SOFTKEYHANDLE CCAUXDLL_CALLING_CONV CrossControl::GetSoftKey (
    void )
```

Factory function that creates instances of the SoftKey object.

Supported Platform(s): VI2



**Returns**

SOFTKEYHANDLE to an allocated SoftKey object. The returned handle needs to be deallocated using the [SoftKey\\_release\(SOFTKEYHANDLE\)](#) method when it's no longer needed.

Returns NULL if it fails to allocate memory.

**Example Usage:**

```
SOFTKEYHANDLE pSoftKey = ::GetSoftKey();
assert(pSoftKey);
softkey_example(pSoftKey);
SoftKey_release(pSoftKey);
```

**SoftKey\_getBacklightEnabledDuringStartup()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::SoftKey_get↔
BacklightEnabledDuringStartup (
    SOFTKEYHANDLE ,
    CCStatus * status )
```

Is the Softkey Backlight enabled during startup? If enabled, the LED will blink to indicate startup progress. It will turn solid once the OS has started.

Supported Platform(s): VI2

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_getBacklightIdleTime()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::SoftKey_get↔
BacklightIdleTime (
    SOFTKEYHANDLE ,
    uint8_t * idleTime )
```

Get SoftKey Backlight IDLE time.

Supported Platform(s): VI2

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_getBacklightIntensity()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey.get←
BacklightIntensity (
    SOFTKEYHANDLE ,
    uint8_t key,
    uint8_t * intensity )
```

Get softkey backlight intensity.

Supported Platform(s): VI2

Parameters

<i>key</i>	Key that's intensity is read. Start index 1.
<i>intensity</i>	Intensity value 0 - 100%

Due limitation in HW, readback value may differ from the value that was set.

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
err = SoftKey_getBacklightIntensity(pSoftKey, 1, &value8);
if (err == CrossControl::ERR_SUCCESS)
    cout << "SoftKey 1 backlight intensity is " << std::dec << value8 <<
        std::endl;
```

**SoftKey\_getBacklightNrOfPulses()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey.get←
BacklightNrOfPulses (
    SOFTKEYHANDLE ,
    uint8_t * nrOfPulses )
```

Get number of pulses during a Softkey backlight blink sequence.

Supported Platform(s): VI2

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_getBacklightOffTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_get↔
BacklightOffTime (
    SOFTKEYHANDLE ,
    uint8_t * offTime )
```

Get SoftKey Backlight OFF time.

Supported Platform(s): VI2

## Parameters

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

## Returns

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

**SoftKey\_getBacklightOnTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_get↔
BacklightOnTime (
    SOFTKEYHANDLE ,
    uint8_t * onTime )
```

Get SoftKey Backlight ON time.

Supported Platform(s): VI2

## Parameters

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

## Returns

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

**SoftKey\_getBacklightSignal()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_get↔
BacklightSignal (
    SOFTKEYHANDLE ,
    float64_t * frequency,
    uint8_t * dutyCycle )
```

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

Supported Platform(s): VI2

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
err = SoftKey_getBacklightSignal(pSoftKey, &fvalue64, &value8);
cout.precision(1);
if (err == CrossControl::ERR_SUCCESS)
    cout << "SoftKey backlight signal frequency is "<< std::fixed << fvalue64 << "
        and duty cycle " << std::dec << value8 << std::endl;
```

**SoftKey\_getBootBacklightConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_get↔
BootBacklightConfig (
    SOFTKEYHANDLE ,
    uint8_t * bootIntensity,
    float32_t * bootFrequency,
    uint8_t * bootDutyCycle,
    uint8_t * postBootIntensity,
    float32_t * postBootFrequency,
    uint8_t * postBootDutyCycle,
    CCStatus * postBootConfig )
```

Get SoftKey backlight configuration during and after system boot. These settings applies to all SoftKeys, they cannot be configured individually. The behavior of the backlight LED in terms of intensity and blink frequency may be configured for system boot and after boot. The post-boot setting can also be disabled using the postBoot↔Config parameter. When disabled, the boot-settings will remain until a user application reconfigures the backlight.

Supported Platform(s): VI2

## Parameters

<i>bootIntensity</i>	Backlight intensity (0-100%) during system boot.
<i>bootFrequency</i>	Backlight blink frequency (0.2-50 Hz) during system boot.
<i>bootDutyCycle</i>	Backlight on duty cycle (0-100%) during system boot.
<i>postBootIntensity</i>	Backlight intensity (0-100%) after system boot.
<i>postBootFrequency</i>	Backlight blink frequency (0.2-50 Hz) after system boot.
<i>postBootDutyCycle</i>	Backlight on duty cycle (0-100%) after system boot.
<i>postBootConfig</i>	Post-boot settings enabled or disabled.

Due limitation in HW, readback value may differ from the value that was set.

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_getMultipleBacklightIntensities()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_get↔
MultipleBacklightIntensities (
    SOFTKEYHANDLE ,
    uint8_t * keys,
    uint8_t * intensities,
    uint8_t array_size )
```

Get multiple SoftKey backlight intensities.  
Supported Platform(s): VI2

**Parameters**

<i>keys</i>	Array of Keys that's intensity to set. Key indexing starts at number 1.
<i>intensities</i>	Array of intensities for the keys listed above. Inetnsity value in range 0 - 100%
<i>array_size</i>	Number of key/intensity - pairs

Due limitation in HW, readback value may differ from the value that was set.

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
keys[0] = 1;
keys[1] = 2;
keys[2] = 3;
keys[3] = 4;
memset(intensities, 0, sizeof(intensities));
err = SoftKey_getMultipleBacklightIntensities(pSoftKey, keys, intensities,
    sizeof(keys));
if (err == CrossControl::ERR_SUCCESS) {
    cout << "SoftKey get multiple keys intensities" << std::endl;
    cout << "SoftKey 1 intensity: " << +intensities[0] << std::endl;
    cout << "SoftKey 2 intensity: " << +intensities[1] << std::endl;
    cout << "SoftKey 3 intensity: " << +intensities[2] << std::endl;
    cout << "SoftKey 4 intensity: " << +intensities[3] << std::endl;
}
```

**SoftKey\_getStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_get↔
Status (
    SOFTKEYHANDLE ,
    uint16_t * value )
```

Get current status of keys

Supported Platform(s): VI2

## Parameters

<i>value</i>	Bitfield of keys that are pressed. LSB is KEY1. Can be undefined if return value is error code.
--------------	---

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
err = SoftKey.getStatus(pSoftKey, &value16);
if (err == CrossControl::ERR_SUCCESS) {
    std::bitset<16> status(value16);
    cout << "SoftKey status " << status << std::endl;
}
```

**SoftKey\_release()**

```
EXTERN_C CCAUXDLLAPI void CCAUXDLLCALLING_CONV CrossControl::SoftKey_release
(
```

```
    SOFTKEYHANDLE )
```

Delete the SOFTKEY object.

Supported Platform(s): VI2

## Returns

-

## Example Usage:

```
SOFTKEYHANDLE pSoftKey = ::GetSoftKey();
assert(pSoftKey);
softkey_example(pSoftKey);
SoftKey_release(pSoftKey);
```

**SoftKey\_setBacklightEnabledDuringStartup()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::SoftKey_set↔
BacklightEnabledDuringStartup (
```

```
    SOFTKEYHANDLE ,
```

```
    CCStatus status )
```

Should the Softkey Backlight be enabled during startup? If enabled, the LED will blink to indicate startup progress. It will turn solid once the OS has started.

Supported Platform(s): VI2

## Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_setBacklightIdleTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↔
BacklightIdleTime (
```

```
    SOFTKEYHANDLE ,
    uint8_t idleTime )
```

Set SoftKey Backlight IDLE time.

Supported Platform(s): VI2

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_setBacklightIntensity()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↔
BacklightIntensity (
```

```
    SOFTKEYHANDLE ,
    uint8_t key,
    uint8_t intensity )
```

Set softkey backlight intensity.

Supported Platform(s): VI2

**Parameters**

<i>key</i>	Key that's intensity is set. Start index 1.
<i>intensity</i>	Intensity value in range 0 - 100%

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = SoftKey_setBacklightIntensity(pSoftKey, 1, 50);
if (err == CrossControl::ERR_SUCCESS)
    cout << "SoftKey 1 backlight set to 50" << std::endl;
```



**SoftKey\_setBacklightNrOfPulses()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↔
BacklightNrOfPulses (
    SOFTKEYHANDLE ,
    uint8_t nrOfPulses )
```

Set number of pulses during a SoftKey Backlight blink sequence.  
Supported Platform(s): VI2

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_setBacklightOff()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↔
BacklightOff (
    SOFTKEYHANDLE )
```

Set Softkey Backlight off.  
Supported Platform(s): VI2

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_setBacklightOffTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↔
BacklightOffTime (
    SOFTKEYHANDLE ,
    uint8_t offTime )
```

Set SoftKey Backlight OFF time.  
Supported Platform(s): VI2

## Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = SoftKey.setBacklightOffTime(pSoftKey, 50);
if (err == CrossControl::ERR_SUCCESS)
    cout << "SoftKey backlight OffTime set to 50" << std::endl;
```

**SoftKey.setBacklightOnTime()**

EXTERN\_C CCAUXDLLAPI eErr CCAUXDLLCALLING\_CONV CrossControl::SoftKey.set↵  
BacklightOnTime (

SOFTKEYHANDLE ,  
uint8\_t onTime )

Set SoftKey Backlight ON time.

Supported Platform(s): VI2

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = SoftKey.setBacklightOnTime(pSoftKey, 50);
if (err == CrossControl::ERR_SUCCESS)
    cout << "SoftKey backlight OnTime set to 50" << std::endl;
```

**SoftKey.setBacklightSignal()**

EXTERN\_C CCAUXDLLAPI eErr CCAUXDLLCALLING\_CONV CrossControl::SoftKey.set↵  
BacklightSignal (

SOFTKEYHANDLE ,  
float64\_t frequency,  
uint8\_t dutyCycle )

Set SoftKey Backlight signal.

Supported Platform(s): VI2

**Parameters**

<i>frequency</i>	SoftKey Backlight blink frequency (0.2-50 Hz).
<i>dutyCycle</i>	SoftKey Backlight ON duty cycle (0-100%).

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Note: The hardware cannot be set to have an on or off time of the LED that's longer than 2.55s (255\*10ms) Hence there are limitations in this function when using frequencies slower than 0.39Hz. At 0.38Hz, the valid duty cycle range is [3 - 97]. At 0.30Hz, the valid duty cycle range is [24 - 76]. At 0.20Hz, the valid duty cycle range is [49 - 51]. At 0.19Hz and slower, the behavior is undefined for all duty cycles, so this is not allowed to be set. Additionally, the hardware cannot be set to have an on or off time of the LED that's shorter than 10ms. Hence, there are limitations in this function when using high frequencies. At 50 Hz, the valid duty cycle range is [50]. At 30 Hz, the valid duty cycle range is [30-70]. At 10 Hz, the valid duty cycle range is [10-90]. At 2 Hz, the valid duty cycle range is [2-98]. The behavior is undefined outside these ranges but setting 0% or 100% duty cycle will always work, regardless of the frequency. If you need to blink in an unsupported range, it can be done with a software timer instead.

## Example Usage:

```
err = SoftKey_setBacklightSignal(pSoftKey, 2, 50);
if (err == CrossControl::ERR_SUCCESS)
    cout << "SoftKey backlight signal set to 2Hz 50% duty cycle" << std::endl;
```

**SoftKey\_setBootBacklightConfig()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↵
BootBacklightConfig (
    SOFTKEYHANDLE ,
    uint8_t bootIntensity,
    float32_t bootFrequency,
    uint8_t bootDutyCycle,
    uint8_t postBootIntensity,
    float32_t postBootFrequency,
    uint8_t postBootDutyCycle,
    CCStatus postBootConfig )
```

Set SoftKey backlight configuration during and after system boot. These settings applies to all SoftKeys, they cannot be configured individually. The behavior of the backlight LED in terms of intensity and blink frequency may be configured for system boot and after boot. The post-boot setting can also be disabled using the postBoot↵ Config parameter. When disabled, the boot-settings will remain until a user application reconfigures the backlight. Note that the frequency limitations mentioned in the description of SoftKey\_setBacklightSignal also apply here.

Supported Platform(s): VI2

## Parameters

<i>bootIntensity</i>	Backlight intensity (0-100%) during system boot.
<i>bootFrequency</i>	Backlight blink frequency (0.2-50 Hz) during system boot.
<i>bootDutyCycle</i>	Backlight on duty cycle (0-100%) during system boot.
<i>postBootIntensity</i>	Backlight intensity (0-100%) after system boot.
<i>postBootFrequency</i>	Backlight blink frequency (0.2-50 Hz) after system boot.

## Parameters

<i>postBootDutyCycle</i>	Backlight on duty cycle (0-100%) after system boot.
<i>postBootConfig</i>	Post-boot settings enabled or disabled.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**SoftKey\_setMultipleBacklightIntensities()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_setMultipleBacklightIntensities (
    SOFTKEYHANDLE ,
    uint8_t * keys,
    uint8_t * intensities,
    uint8_t array_size )
```

Set Multiple SoftKey backlight intensities.

Supported Platform(s): VI2

## Parameters

<i>keys</i>	Array of Keys that's intensity to set. Key indexing starts at number 1.
<i>intensities</i>	Array of intensities for the keys listed above. Intensity value in range 0 - 100%
<i>array_size</i>	Number of key/intensity - pairs

To set all keys to same intensity. Use key number 0 and array\_size 1.

Not all values are supported on all platforms, ERR\_NOT\_SUPPORTED will indicate that.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
keys[0] = 1;
keys[1] = 4;
keys[2] = 2;
keys[3] = 3;
intensities[0] = 25;
intensities[1] = 50;
intensities[2] = 75;
intensities[3] = 100;
err = SoftKey_setMultipleBacklightIntensities(pSoftKey, keys, intensities,
    sizeof(keys));
if (err == CrossControl::ERR_SUCCESS) {
    cout << "SoftKey set multiple keys intensities" << std::endl;
    cout << "SoftKey 1 intensity: " << +intensities[0] << std::endl;
    cout << "SoftKey 2 intensity: " << +intensities[3] << std::endl;
    cout << "SoftKey 3 intensity: " << +intensities[2] << std::endl;
```

```

    cout << "SoftKey 4 intensity: " << +intensities[1] << std::endl;
}

```

## 19.20 Telematics functions

### Functions

- EXTERN\_C CCAUXDLL\_API [TELEMATICSHANDLE](#) CCAUXDLL\_CALLING\_CONV \_CONV [GetTelematics](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Telematics\\_release](#) ([TELEMATICSHANDLE](#))
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getTelematicsAvailable](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getGPRSPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getGPRSStartupPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getWLANPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getWLANStartupPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getBTPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getBTStartupPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getGPSPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getGPSStartupPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_getGPSAntennaStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_setGPRSPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_setGPRSStartupPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_setWLANPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_setWLANStartupPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_setBTPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_setBTStartupPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_setGPSPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [Telematics\\_setGPSStartupPowerStatus](#) ([TELEMATICSHANDLE](#), [CCStatus](#) status)

## 19.20.1 Detailed Description

Functions in the Telematics class

## 19.20.2 Function Documentation

### GetTelematics()

```
EXTERN_C CCAUXDLL_API TELEMATICSHANDLE CCAUXDLL_CALLING_CONV CrossControl↔
::GetTelematics (
    void )
```

Factory function that creates instances of the Telematics object.  
Supported Platform(s): XM, XA, XS

#### Returns

[TELEMATICSHANDLE](#) to an allocated Telematics object. The returned handle needs to be deallocated using the [Telematics\\_release\(TELEMATICSHANDLE\)](#) 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);
Telematics\_release(pTelematics);
```

### Telematics\_getBTPowerStatus()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_getBTPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )
```

Get Bluetooth power status.  
Supported Platform(s): XM, XA, XS

#### Parameters

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

#### Returns

error status. 0 = [ERR\\_SUCCESS](#), otherwise error code. See the enum [eErr](#) for details.

#### Example Usage:

```
err = Telematics\_getBTPowerStatus(pTelematics, &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;
}

```

### Telematics\_getBTStartupPowerStatus()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↵
_getBTStartupPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )

```

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

Supported Platform(s): XM, XA, XS

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

#### Example Usage:

```

err = Telematics_getBTStartupPowerStatus(pTelematics, &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;
}

```

### Telematics\_getGPRSPowerStatus()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↵
_getGPRSPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )

```

Get GPRS power status.

Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
err = Telematics.getGPRSPowerStatus(pTelematics, &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;
}
```

**Telematics.getGPRSStartupPowerStatus()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Telematics←
_getGPRSStartupPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )
```

Get GPRS power status at startup and at resume from suspended mode.  
Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

## Example Usage:

```
err = Telematics.getGPRSStartupPowerStatus(pTelematics, &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;
}

```

### Telematics\_getGPSAntennaStatus()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_getGPSAntennaStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )

```

Get GPS antenna status. Antenna open/short detection. The status is set to disabled if no antenna is present or a short is detected. Note, This function is only supported on revision A Telematic Addon Cards (produced before 2015-09).

Supported Platform(s): XM

#### Parameters

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

#### Returns

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

#### Example Usage:

```

err = Telematics.getGPSAntennaStatus(pTelematics, &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;
}

```

### Telematics\_getGPSPowerStatus()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_getGPSPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )

```

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

Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

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

## Example Usage:

```
err = Telematics.getGPSPowerStatus(pTelematics, &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;
}
```

**Telematics.getGPSStartupPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
.getGPSStartupPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )
```

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

Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

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

## Example Usage:

```
err = Telematics.getGPSStartupPowerStatus(pTelematics, &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;
}

```

### Telematics\_getTelematicsAvailable()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_getTelematicsAvailable (
    TELEMATICSHANDLE ,
    CCStatus * status )

```

Is a telematics add-on card installed?

Supported Platform(s): XM, XA, XS

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

#### Example Usage:

```

err = Telematics_getTelematicsAvailable(pTelematics, &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;
}

```

### Telematics\_getWLANPowerStatus()

```

EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_getWLANPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )

```

Get WLAN power status.

Supported Platform(s): XM, XA, XS

#### Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = Telematics_getWLANPowerStatus(pTelematics, &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;
}
```

**Telematics\_getWLANStartupPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_getWLANStartupPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus * status )
```

Get WLAN power status at startup and at resume from suspended mode.  
Supported Platform(s): XM, XA, XS

**Parameters**

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Example Usage:**

```
err = Telematics_getWLANStartupPowerStatus(pTelematics, &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;
}
```

**Telematics\_release()**

```
EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_release (
```

```
    TELEMATICSHANDLE )
```

Delete the Telematics object.

Supported Platform(s): XM, XA, XS

## Returns

-

## Example Usage:

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

**Telematics\_setBTPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_setBTPowerStatus (
```

```
    TELEMATICSHANDLE ,
```

```
    CCStatus status )
```

Set Bluetooth power status.

Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Telematics\_setBTStartupPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_setBTStartupPowerStatus (
```

```
    TELEMATICSHANDLE ,
```

```
    CCStatus status )
```

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

Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Telematics\_setGPRSPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_setGPRSPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus status )
```

Set GPRS modem power status.  
Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Telematics\_setGPRSStartupPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_setGPRSStartupPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus status )
```

Set GPRS power status at startup and at resume from suspended mode.  
Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Telematics\_setGPSPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
```

```

_setGPSPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus status )

```

Set GPS power status.

Supported Platform(s): XM, XA, XS

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Telematics\_setGPSStartupPowerStatus()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_setGPSStartupPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus status )

```

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

Supported Platform(s): XM, XA, XS

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Telematics\_setWLANPowerStatus()

```

EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_setWLANPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus status )

```

Set WLAN power status.

Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

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

**Telematics\_setWLANStartUpPowerStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↔
_setWLANStartUpPowerStatus (
    TELEMATICSHANDLE ,
    CCStatus status )
```

Set WLAN power status at startup and at resume from suspended mode.  
Supported Platform(s): XM, XA, XS

## Parameters

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

## Returns

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

**19.21 TouchScreen functions****Functions**

- EXTERN\_C CCAUXDLL\_API [TOUCHSCREENHANDLE](#) CCAUXDLL\_CALLING\_CONV↔  
\_CONV [GetTouchScreen](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [TouchScreen\\_release](#)  
([TOUCHSCREENHANDLE](#))
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getMode](#)  
([TOUCHSCREENHANDLE](#), [TouchScreenModeSettings](#) \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getMouseRightClickTime](#)  
([TOUCHSCREENHANDLE](#), uint16\_t \*time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setMode](#)  
([TOUCHSCREENHANDLE](#), [TouchScreenModeSettings](#) config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setMouseRightClickTime](#)  
([TOUCHSCREENHANDLE](#), uint16\_t time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setAdvancedSetting](#)  
([TOUCHSCREENHANDLE](#), [TSAAdvancedSettingsParameter](#) param, uint16\_t data)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getAdvancedSetting](#)  
([TOUCHSCREENHANDLE](#), [TSAAdvancedSettingsParameter](#) param, uint16\_t \*data)



### 19.21.1 Detailed Description

Functions in the TouchScreen class

### 19.21.2 Function Documentation

#### GetTouchScreen()

```
EXTERN_C CCAUXDLL_API TOUCHSCREENHANDLE CCAUXDLL_CALLING_CONV CrossControl↵
::GetTouchScreen (
    void )
```

Factory function that creates instances of the TouchScreen object.

Supported Platform(s): XL, XM, XS, XA

#### Returns

TOUCHSCREENHANDLE to an allocated TouchScreen object. The returned handle needs to be deallocated using the [TouchScreen\\_release\(TOUCHSCREENHANDLE\)](#) 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);
TouchScreen_release(pTouchScreen);
```

#### TouchScreen\_getAdvancedSetting()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↵
_getAdvancedSetting (
    TOUCHSCREENHANDLE ,
    TSAdvancedSettingsParameter param,
    uint16_t * data )
```

Get advanced touch screen settings. See the description of TSAdvancedSettings↵ Parameter for a description of the parameters.

Supported Platform(s): XL, XM, XS, XA

#### 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. See the enum eErr for details.

#### Example Usage:

```
err = TouchScreen_getAdvancedSetting(pTouchScreen, 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;
}

```

### TouchScreen\_getMode()

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::TouchScreen←  
\_getMode (

TOUCHSCREENHANDLE ,  
TouchScreenModeSettings \* config )

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

Supported Platform(s): XL, XM, XS, XA

#### Parameters

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

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

#### Example Usage:

```

err = TouchScreen_getMode(pTouchScreen, &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 if (err == ERR_NOT_SUPPORTED)
{
    cout << "Function TouchScreen_getMode() is not supported on this
        platform" << endl;
}
else
{
    cout << "Error(" << err << ") in function getMode: " << GetErrorStringA(err)
        << endl;
}

```

### TouchScreen\_getMouseRightClickTime()

EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CrossControl::TouchScreen←  
\_getMouseRightClickTime (

```
TOUCHSCREENHANDLE ,
uint16_t * time )
```

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

Supported Platform(s): XL, XM, XS, XA

Parameters

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

Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

Example Usage:

```
err = TouchScreen.getMouseRightClickTime(pTouchScreen, &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;
}
```

**TouchScreen\_release()**

```
EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↵
_release (
```

```
TOUCHSCREENHANDLE )
```

Delete the TouchScreen object.

Supported Platform(s): XL, XM, XS, XA

Returns

-

Example Usage:

```
TOUCHSCREENHANDLE pTouchScreen = ::GetTouchScreen();
assert(pTouchScreen);
touchscreen.example(pTouchScreen);
TouchScreen.release(pTouchScreen);
```

**TouchScreen\_setAdvancedSetting()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↵
_setAdvancedSetting (
```

```
TOUCHSCREENHANDLE ,
```

```
TSAdvancedSettingsParameter param,
```

```
uint16_t data )
```

Set advanced touch screen settings. See the description of TSAdvancedSettings↵ Parameter for a description of the parameters.

Supported Platform(s): XL, XM, XS, XA

## 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. See the enum eErr for details.

**TouchScreen\_setMode()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↔
_setMode (
    TOUCHSCREENHANDLE ,
    TouchScreenModeSettings config )
```

Set Touch Screen mode. Sets the mode of the USB profile.  
Supported Platform(s): XL, XM, XS, XA

## Parameters

<i>config</i>	The mode to set.
---------------	------------------

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**TouchScreen\_setMouseRightClickTime()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↔
_setMouseRightClickTime (
    TOUCHSCREENHANDLE ,
    uint16_t time )
```

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

Supported Platform(s): XL, XM, XS, XA

## Parameters

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

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**19.22 TouchScreenCalib functions****Functions**

- EXTERN\_C CCAUXDLL\_API TOUCHSCREENCALIBHANDLE CCAUXDLL↔  
\_CALLING\_CONV GetTouchScreenCalib (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV TouchScreenCalib\_release  
(TOUCHSCREENCALIBHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV TouchScreenCalib\_setMode  
(TOUCHSCREENCALIBHANDLE, CalibrationModeSettings mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV TouchScreenCalib\_getMode  
(TOUCHSCREENCALIBHANDLE, CalibrationModeSettings \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV TouchScreenCalib\_setCalibrationPoint  
(TOUCHSCREENCALIBHANDLE, uint8\_t pointNr)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV TouchScreenCalib\_checkCalibrationPoint  
(TOUCHSCREENCALIBHANDLE, bool \*finished, uint8\_t pointNr)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV TouchScreenCalib\_getConfigParam  
(TOUCHSCREENCALIBHANDLE, CalibrationConfigParam param, uint16\_t  
\*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV TouchScreenCalib\_setConfigParam  
(TOUCHSCREENCALIBHANDLE, CalibrationConfigParam param, uint16\_t  
value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV TouchScreenCalib\_autoSensorCalib  
(TOUCHSCREENCALIBHANDLE)

**19.22.1 Detailed Description**

Functions in the TouchScreenCalib class

**19.22.2 Function Documentation****GetTouchScreenCalib()**

```
EXTERN_C CCAUXDLL_API TOUCHSCREENCALIBHANDLE CCAUXDLL_CALLING_CONV Cross↔
Control::GetTouchScreenCalib (
    void )
```

Factory function that creates instances of the TouchScreenCalib object.

Supported Platform(s): XL, XM, XS, XA

## Returns

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

**TouchScreenCalib\_autoSensorCalib()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↵
Calib_autoSensorCalib (
    TOUCHSCREENCALIBHANDLE )
```

Perform automatic sensor calibration  
Supported Platform(s): VA

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code.

**TouchScreenCalib\_checkCalibrationPointFinished()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↵
Calib_checkCalibrationPointFinished (
    TOUCHSCREENCALIBHANDLE ,
    bool * finished,
    uint8_t pointNr )
```

Check if a calibration point is finished  
Supported Platform(s): XL, XM, XS, XA

## 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.

**TouchScreenCalib\_getConfigParam()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↵
Calib_getConfigParam (
    TOUCHSCREENCALIBHANDLE ,
    CalibrationConfigParam param,
    uint16_t * value )
```

Get calibration config parameters  
Supported Platform(s): XL, XM, XS, XA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code.

**TouchScreenCalib\_getMode()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↔
Calib_getMode (
    TOUCHSCREENCALIBHANDLE ,
    CalibrationModeSettings * mode )
```

Get mode of front controller.

Supported Platform(s): XL, XM, XS, XA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code.

**TouchScreenCalib\_release()**

```
EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↔
Calib_release (
    TOUCHSCREENCALIBHANDLE )
```

Delete the TouchScreenCalib object.

Supported Platform(s): XL, XM, XS, XA

## Returns

-

**TouchScreenCalib\_setCalibrationPoint()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↔
Calib_setCalibrationPoint (
    TOUCHSCREENCALIBHANDLE ,
    uint8_t pointNr )
```

Set calibration point

Supported Platform(s): XL, XM, XS, XA



## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code.

**TouchScreenCalib\_setConfigParam()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::TouchScreen↔
Calib_setConfigParam (
    TOUCHSCREENCALIBHANDLE ,
    CalibrationConfigParam param,
    uint16_t value )
```

Set calibration config parameters

Supported Platform(s): XL, XM, XS, XA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code.

**TouchScreenCalib\_setMode()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::TouchScreen↔
Calib_setMode (
    TOUCHSCREENCALIBHANDLE ,
    CalibrationModeSettings mode )
```

Set mode of front controller.

Supported Platform(s): XL, XM, XS, XA

## Parameters

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

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code.

## 19.23 Video functions

### Functions

- EXTERN\_C CCAUXDLL\_API VIDEOHANDLE CCAUXDLL\_CALLING\_CONV GetVideo (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Video\_release (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_init (VIDEOHANDLE, uint8\_t deviceNr)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_showVideo (VIDEOHANDLE, bool show)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setDeInterlaceMode (VIDEOHANDLE, DeInterlaceMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getDeInterlaceMode (VIDEOHANDLE, DeInterlaceMode \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setMirroring (VIDEOHANDLE, CCStatus mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getMirroring (VIDEOHANDLE, CCStatus \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setRotation (VIDEOHANDLE, VideoRotation rotation)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getRotation (VIDEOHANDLE, VideoRotation \*rotation)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setActiveChannel (VIDEOHANDLE, VideoChannel channel)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getActiveChannel (VIDEOHANDLE, VideoChannel \*channel)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setColorKeys (VIDEOHANDLE, uint8\_t rKey, uint8\_t gKey, uint8\_t bKey)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getColorKeys (VIDEOHANDLE, uint8\_t \*rKey, uint8\_t \*gKey, uint8\_t \*bKey)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setVideoArea (VIDEOHANDLE, uint16\_t topLeftX, uint16\_t topLeftY, uint16\_t bottomRightX, uint16\_t bottomRightY)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getRawImage (VIDEOHANDLE, uint16\_t \*width, uint16\_t \*height, float32\_t \*frameRate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getVideoArea (VIDEOHANDLE, uint16\_t \*topLeftX, uint16\_t \*topLeftY, uint16\_t \*bottomRightX, uint16\_t \*bottomRightY)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getVideoStandard (VIDEOHANDLE, videoStandard \*standard)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getStatus (VIDEOHANDLE, uint8\_t \*status)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setScaling (VIDEOHANDLE, float32\_t x, float32\_t y)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getScaling (VIDEOHANDLE, float32\_t \*x, float32\_t \*y)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_activateSnapshot (VIDEOHANDLE, bool activate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_takeSnapshot (VIDEOHANDLE, const char\_t \*path, bool bInterlaced)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_takeSnapshotRaw (VIDEOHANDLE, char\_t \*rawImgBuffer, uint32\_t rawImgBufSize, bool bInterlaced)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_takeSnapshotBmp (VIDEOHANDLE, char\_t \*\*bmpBuffer, uint32\_t \*bmpBufSize, bool bInterlaced, bool bNTSCFormat)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_createBitmap (VIDEOHANDLE, char\_t \*\*bmpBuffer, uint32\_t \*bmpBufSize, const char\_t \*rawImgBuffer, uint32\_t rawImgBufSize, bool bInterlaced, bool bNTSCFormat)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_freeBmpBuffer (VIDEOHANDLE, char\_t \*bmpBuffer)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_minimize (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_restore (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setDecoderReg (VIDEOHANDLE, uint8\_t decoderRegister, uint8\_t registerValue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getDecoderReg (VIDEOHANDLE, uint8\_t decoderRegister, uint8\_t \*registerValue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setCropping (VIDEOHANDLE, uint8\_t top, uint8\_t left, uint8\_t bottom, uint8\_t right)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getCropping (VIDEOHANDLE, uint8\_t \*top, uint8\_t \*left, uint8\_t \*bottom, uint8\_t \*right)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_showFrame (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setGraphicsOverlay (VIDEOHANDLE, CCStatus mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getGraphicsOverlay (VIDEOHANDLE, CCStatus \*mode)

### 19.23.1 Detailed Description

Functions in the Video class

### 19.23.2 Function Documentation

**GetVideo()**

```
EXTERN_C CCAUXDLL_API VIDEOHANDLE CCAUXDLL_CALLING_CONV CrossControl::GetVideo (
    void )
```

Factory function that creates instances of the Video object.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

**Returns**

VIDEOHANDLE to an allocated Video object. The returned handle needs to be deallocated using the [Video\\_release\(VIDEOHANDLE\)](#) method when it's no longer needed. Returns NULL if it fails to allocate memory.

**Video\_activateSnapshot()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_activateSnapshot (
    VIDEOHANDLE ,
    bool activate )
```

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.

Supported Platform(s): XL, XM (Windows)

**Parameters**

<i>activate</i>	Set to true if the snapshot function shall be active.
-----------------	---

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_createBitmap()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_createBitmap (
    VIDEOHANDLE ,
    char_t ** bmpBuffer,
    uint32_t * bmpBufSize,
    const char_t * rawImgBuffer,
    uint32_t rawImgBufSize,
    bool bInterlaced,
    bool bNTSCFormat )
```

Create a bitmap from a raw image buffer. The bmp buffer is allocated in the function and has to be deallocated by the application.

Supported Platform(s): XL, XM (Windows)

#### Parameters

<i>bmpBuffer</i>	Bitmap ram buffer allocated by the API, has to be deallocated with <code>freeBmpBuffer()</code> by the application.
<i>bmpBufSize</i>	Size of the returned bitmap buffer.
<i>rawImgBuffer</i>	Raw image buffer from <code>takeSnapshotRaw</code> .
<i>rawImgBufSize</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>bNTSCFormat</i>	True if the video format in <code>rawImageBuffer</code> is NTSC format.

#### Returns

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

### Video\_freeBmpBuffer()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_freeBmpBuffer (
    VIDEOHANDLE ,
    char_t * bmpBuffer )
```

Free the memory allocated for BMP buffer.

Supported Platform(s): XL, XM (Windows)

#### Parameters

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

#### Returns

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

### Video\_getActiveChannel()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_getActiveChannel (
    VIDEOHANDLE ,
    VideoChannel * channel )
```

Get the current video channel.

Supported Platform(s): XL, XM, XS, XA, VC, VA

## Parameters

<i>channel</i>	Enum defining available channels. (VC platform has only 1 channel, Analog_Channel_1)
----------------	--

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_getColorKeys()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_getColorKeys (
    VIDEOHANDLE ,
    uint8_t * rKey,
    uint8_t * gKey,
    uint8_t * bKey )
```

Get color key values. Note that the system uses 18 bit colors, so the two least significant bits are not used.

Supported Platform(s): XL, XM

## 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. See the enum eErr for details.

**Video\_getCropping()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_getCropping (
    VIDEOHANDLE ,
    uint8_t * top,
    uint8_t * left,
    uint8_t * bottom,
    uint8_t * right )
```

Get Crop parameters.

Supported Platform(s): XL, XM, XS, XA, VC, VA

## 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. See the enum eErr for details.

**Video\_getDecoderReg()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getDecoderReg (
    VIDEOHANDLE ,
    uint8_t decoderRegister,
    uint8_t * registerValue )
```

Get Video decoder bus register. Advanced function for direct access to the video decoder TVP5150AM1 registers.

Supported Platform(s): XL, XM, XS, XA, VC, VA

## Parameters

<i>decoderRegister</i>	Decoder Register Address.
<i>registerValue</i>	register value.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_getDeInterlaceMode()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getDeInterlaceMode (
    VIDEOHANDLE ,
    DeInterlaceMode * mode )
```

Get the deinterlace mode used when decoding the interlaced video stream.

Supported Platform(s): XL, XM

## Parameters

<i>mode</i>	The current mode. See enum DeInterlaceMode for descriptions of the modes.
-------------	---

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_getGraphicsOverlay()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getGraphicsOverlay (
```

```
    VIDEOHANDLE ,  
    CCStatus * mode )
```

Get the current graphics overlaying mode.  
Supported Platform(s): XA, XS, VC, VA

## Parameters

<i>mode</i>	Overlay enable mode
-------------	---------------------

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_getMirroring()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getMirroring (
```

```
    VIDEOHANDLE ,  
    CCStatus * mode )
```

Get the current mirroring mode of the video image.  
Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## Parameters

<i>mode</i>	The current mode. Enabled or Disabled.
-------------	--

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.



**Video\_getRawImage()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_getRawImage (
```

```
    VIDEOHANDLE ,
    uint16_t * width,
    uint16_t * height,
    float32_t * frameRate )
```

Get the raw image size of moving image before any scaling and frame rate. For snapshot the height is 4 row less.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## 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. See the enum eErr for details.

**Video\_getRotation()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_getRotation (
```

```
    VIDEOHANDLE ,
    VideoRotation * rotation )
```

Get the current rotation of the video image.

Supported Platform(s): XA, XS, VC, VA, VS

## Parameters

<i>rotation</i>	Enum defining the current rotation.
-----------------	-------------------------------------

## Returns

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

**Video\_getScaling()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getScaling
(
    VIDEOHANDLE ,
    float32_t * x,
    float32_t * y )
```

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.

Supported Platform(s): XL, XM

## 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. See the enum eErr for details.

**Video\_getStatus()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getStatus
(
    VIDEOHANDLE ,
    uint8_t * status )
```

Video status byte.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## 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. See the enum eErr for details.

**Video\_getVideoArea()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getVideoArea (
    VIDEOHANDLE ,
    uint16_t * topLeftX,
    uint16_t * topLeftY,
    uint16_t * bottomRigthX,
    uint16_t * bottomRigthY )
```

Get the area where video is shown.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## Parameters

<i>topLeftX</i>	Top left X coordinate on screen.
<i>topLeftY</i>	Top left Y coordinate on screen.
<i>bottomRigthX</i>	Bottom right X coordinate on screen.
<i>bottomRigthY</i>	Bottom right Y coordinate on screen.

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_getVideoStandard()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getVideoStandard (
    VIDEOHANDLE ,
    videoStandard * standard )
```

Get video standard. The video decoder auto detects the video standard of the source.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## Parameters

<i>standard</i>	Video standard.
-----------------	-----------------

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_init()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_init (
    VIDEOHANDLE ,
    uint8_t deviceNr )
```

Initialize a video device. The video device will initially use the following settings: DeInterlace\_BOB and mirroring disabled.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## Parameters

<i>deviceNr</i>	Device to connect to (1,2). Select one of 2 devices to connect to. (VC platform has only 1 device)
-----------------	--

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_minimize()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_minimize
(
    VIDEOHANDLE )
```

Minimizes the video area. Restore with restore() call.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_release()**

```
EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::Video_release
(
    VIDEOHANDLE )
```

Delete the Video object.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## Returns

-

**Video\_restore()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_restore
(
    VIDEOHANDLE )
```

Restores the video area to the size it was before a minimize() call. Don't use restore if minimize has not been used first.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_setActiveChannel()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_setActive↵
Channel (
    VIDEOHANDLE ,
    VideoChannel channel )
```

Sets the active video channel.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

**Parameters**

<i>channel</i>	Enum defining available channels. (VC and VS platforms have only 1 channel, Analog_Channel_1)
----------------	---

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_setColorKeys()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_setColor↵
Keys (
    VIDEOHANDLE ,
    uint8_t rKey,
    uint8_t gKey,
    uint8_t bKey )
```

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.

Supported Platform(s): XL, XM

## 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. See the enum eErr for details.

**Video\_setCropping()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_setCropping
(
    VIDEOHANDLE ,
    uint8_t top,
    uint8_t left,
    uint8_t bottom,
    uint8_t right )
```

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. On XA/XS platforms, cropping from top/bottom on device 2 (channels 3 and 4) is not supported.

Supported Platform(s): XL, XM, XS, XA, VC, VA

## 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. See the enum eErr for details.

**Video\_setDecoderReg()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_setDecoder←
Reg (
    VIDEOHANDLE ,
```

```
uint8_t decoderRegister,
uint8_t registerValue )
```

Set Video decoder bus register. Advanced function for direct access to the video decoder TVP5150AM1 registers.

Supported Platform(s): XL, XM, XS, XA, VC, VA

#### Parameters

<i>decoderRegister</i>	Decoder Register Address.
<i>registerValue</i>	register value.

#### Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

### Video\_setDeInterlaceMode()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_setDeInterlaceMode (
    VIDEOHANDLE ,
    DeInterlaceMode mode )
```

Set the deinterlace mode used when decoding the interlaced video stream.

Supported Platform(s): XL, XM

#### 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. See the enum eErr for details.

### Video\_setGraphicsOverlay()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_setGraphicsOverlay (
    VIDEOHANDLE ,
    CCStatus mode )
```

Enable or disable overlaying of graphics on top of video.

Supported Platform(s): XA, XS, VC, VA

## Parameters

<i>mode</i>	Overlay enable mode
-------------	---------------------

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_setMirroring()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_setMirroring
(
    VIDEOHANDLE ,
    CCStatus mode )
```

Enable or disable mirroring of the video image.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## Parameters

<i>mode</i>	The mode to set. Enabled or Disabled.
-------------	---------------------------------------

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_setRotation()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLLCALLING_CONV CrossControl::Video_setRotation
(
    VIDEOHANDLE ,
    VideoRotation rotation )
```

Set the current rotation of the video image.

Supported Platform(s): XA, XS, VC, VA, VS

## Parameters

<i>rotation</i>	Enum defining the rotation to set.
-----------------	------------------------------------

## Returns

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.



**Video\_setScaling()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Video_setScaling
(
    VIDEOHANDLE ,
    float32_t x,
    float32_t y )
```

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.

Supported Platform(s): XL, XM

## 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. See the enum eErr for details.

**Video\_setVideoArea()**

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Video_setVideo↵
Area (
    VIDEOHANDLE ,
    uint16_t topLeftX,
    uint16_t topLeftY,
    uint16_t bottomRightX,
    uint16_t bottomRightY )
```

Set the area where video is shown.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

## Parameters

<i>topLeftX</i>	Top left X coordinate on screen.
<i>topLeftY</i>	Top left Y coordinate on screen.
<i>bottomRightX</i>	Bottom right X coordinate on screen.
<i>bottomRightY</i>	Bottom right Y coordinate on screen.

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_showFrame()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_show↵
Frame (
```

```
    VIDEOHANDLE )
```

Copy one frame from camera to the display.

Supported Platform(s): XA, XS, VC, VA

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_showVideo()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_show↵
Video (
```

```
    VIDEOHANDLE ,
```

```
    bool show )
```

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.

Supported Platform(s): XL, XM, XS, XA, VC, VA, VS

**Parameters**

<i>show</i>	True shows the video image.
-------------	-----------------------------

**Returns**

error status. 0 = ERR\_SUCCESS, otherwise error code. See the enum eErr for details.

**Video\_takeSnapshot()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_take↵
Snapshot (
```

```
    VIDEOHANDLE ,
```

```
    const char_t * path,
```

```
    bool bInterlaced )
```

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.

Supported Platform(s): XL, XM (Windows)

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. See the enum `eErr` for details.

### Video.takeSnapshotBmp()

```
EXTERN_C CCAUXDLLAPI eErr CCAUXDLL_CALLING_CONV CrossControl::Video_take←
SnapshotBmp (
    VIDEOHANDLE ,
    char_t ** bmpBuffer,
    uint32_t * bmpBufSize,
    bool bInterlaced,
    bool bNTSCFormat )
```

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.

Supported Platform(s): XL, XM (Windows)

Parameters

<i>bmpBuffer</i>	Bitmap ram buffer allocated by the API, has to be deallocated with <code>freeBmpBuffer()</code> 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>bNTSCFormat</i>	True if the video format in <code>rawImageBuffer</code> is NTSC format.

Returns

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

**Video\_takeSnapshotRaw()**

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL-CALLING-CONV CrossControl::Video_take←
SnapshotRaw (
    VIDEOHANDLE ,
    char_t * rawImgBuffer,
    uint32_t rawImgBuffSize,
    bool bInterlaced )
```

Takes a snapshot of the current video image and return raw image data. The size of the raw image is when interlaced = false  $0x100 + \text{line count} * \text{row count} * 4$ . The size of the raw image is when interlaced = true  $0x100 + \text{line count} * \text{row count} * 2$ . To be able to take a snapshot, the snapshot function has to be active. This function is blocking until a new frame is available from the decoder. An error will be returned if the decoder doesn't return any frames before a timeout.

Supported Platform(s): XL, XM (Windows)

## Parameters

<i>rawImgBuffer</i>	Buffer for image to be stored in.
<i>rawImgBuffSize</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. See the enum eErr for details.

# Chapter 20

## Namespace Documentation

### 20.1 CrossControl Namespace Reference

#### Data Structures

- struct [BatteryTimerType](#)
- struct [received\\_video](#)
- struct [video\\_dec\\_command](#)
- struct [version\\_info](#)
- struct [BuzzerSetup](#)
- struct [LedTimingType](#)
- struct [FpgaLedTimingType](#)
- struct [LedColorMixType](#)
- struct [TimerType](#)
- struct [UpgradeStatus](#)

#### Typedefs

- typedef void \* [ABOUTHANDLE](#)
- typedef void \* [ADCHANDLE](#)
- typedef void \* [AUXVERSIONHANDLE](#)
- typedef void \* [BACKLIGHTHANDLE](#)
- typedef void \* [BATTERYHANDLE](#)
- typedef void \* [BUZZERHANDLE](#)
- typedef void \* [CANSETTINGHANDLE](#)
- typedef struct [version\\_info](#) [VersionType](#)
- typedef void \* [CFGINHANDLE](#)
- typedef void \* [CONFIGHANDLE](#)
- typedef void \* [DIAGNOSTICHANDLE](#)
- typedef void \* [DIGIOHANDLE](#)
- typedef void \* [FIRMWAREUPGHANDLE](#)
- typedef void \* [FRONTLEDHANDLE](#)
- typedef void \* [LIGHTSENSORHANDLE](#)
- typedef void \* [POWERHANDLE](#)

- typedef void \* [POWERMGRHANDLE](#)
- typedef void \* [PWMOUTHANDLE](#)
- typedef void \* [SMARTHANDLE](#)
- typedef void \* [SOFTKEYHANDLE](#)
- typedef void \* [TELEMATICSHANDLE](#)
- typedef void \* [TOUCHSCREENHANDLE](#)
- typedef void \* [TOUCHSCREENCALIBHANDLE](#)
- typedef void \* [VIDEOHANDLE](#)

## 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 }
- enum [VoltageEnum](#) {  
[VOLTAGE\\_24VIN](#) = 0, [VOLTAGE\\_24V](#), [VOLTAGE\\_12V](#), [VOLTAGE\\_12VID](#),  
[VOLTAGE\\_5V](#), [VOLTAGE\\_3V3](#), [VOLTAGE\\_VTFT](#), [VOLTAGE\\_5VSTB](#),  
[VOLTAGE\\_1V9](#), [VOLTAGE\\_1V8](#), [VOLTAGE\\_1V5](#), [VOLTAGE\\_1V2](#),  
[VOLTAGE\\_1V05](#), [VOLTAGE\\_1V0](#), [VOLTAGE\\_0V9](#), [VOLTAGE\\_VREF\\_INT](#),  
[VOLTAGE\\_24V\\_BACKUP](#), [VOLTAGE\\_2V5](#), [VOLTAGE\\_1V1](#), [VOLTAGE\\_1V3\\_PER](#),  
[VOLTAGE\\_1V3\\_VDDA](#), [VOLTAGE\\_3V3STBY](#), [VOLTAGE\\_VPMIC](#), [VOLTAGE\\_VMAIN](#),  
[VOLTAGE\\_UB](#), [VOLTAGE\\_AI1](#), [VOLTAGE\\_AI2](#), [VOLTAGE\\_AI3](#),  
[VOLTAGE\\_5V\\_MP](#), [VOLTAGE\\_12V\\_BL](#), [VOLTAGE\\_VHEAT](#), [VOLTAGE\\_VIN\\_FILT](#),  
[VOLTAGE\\_VIN\\_PROT](#), [VOLTAGE\\_BL\\_ANODE](#), [VOLTAGE\\_IGN\\_STATE](#),  
[VOLTAGE\\_IGN\\_VALUE](#),  
[VOLTAGE\\_PRE\\_IGN\\_STATE](#), [VOLTAGE\\_PRE\\_IGN\\_VALUE](#), [VOLTAGE\\_VBACKUP](#),  
[VOLTAGE\\_END](#) }
- enum [LightSensorOperationRange](#) { [RangeStandard](#) = 0, [RangeExtended](#) = 1 }
- enum [LightSensorSamplingMode](#) { [SamplingModeStandard](#) = 0, [SamplingModeExtended](#), [SamplingModeAuto](#) }
- enum [CCStatus](#) { [Disabled](#) = 0, [Enabled](#) = 1 }
- enum [eErr](#) {  
[ERR\\_SUCCESS](#) = 0, [ERR\\_OPEN\\_FAILED](#) = 1, [ERR\\_NOT\\_SUPPORTED](#) = 2,  
[ERR\\_UNKNOWN\\_FEATURE](#) = 3,  
[ERR\\_DATATYPE\\_MISMATCH](#) = 4, [ERR\\_CODE\\_NOT\\_EXIST](#) = 5, [ERR\\_BUFFER\\_SIZE](#) = 6,  
[ERR\\_IOCTL\\_FAILED](#) = 7,  
[ERR\\_INVALID\\_DATA](#) = 8, [ERR\\_INVALID\\_PARAMETER](#) = 9, [ERR\\_CREATE\\_THREAD](#) = 10,  
[ERR\\_IN\\_PROGRESS](#) = 11,

```

ERR_CHECKSUM = 12 , ERR_INIT_FAILED = 13 , ERR_VERIFY_FAILED =
14 , ERR_DEVICE_READ_DATA_FAILED = 15 ,
ERR_DEVICE_WRITE_DATA_FAILED = 16 , ERR_COMMAND_FAILED =
17 , ERR_EEPROM = 18 , ERR_JIDA_TEMP = 19 ,
ERR_AVERAGE_CALC_STARTED = 20 , ERR_NOT_RUNNING = 21 , ERR_I2C_EXPANDER_READ_FAILED
= 22 , ERR_I2C_EXPANDER_WRITE_FAILED = 23 ,
ERR_I2C_EXPANDER_INIT_FAILED = 24 , ERR_NEWER_SS_VERSION_REQUIRED
= 25 , ERR_NEWER_FPGA_VERSION_REQUIRED = 26 , ERR_NEWER_FRONT_VERSION_REQUIRED
= 27 ,
ERR_TELEMATICS_GPRS_NOT_AVAILABLE = 28 , ERR_TELEMATICS_WLAN_NOT_AVAILABLE
= 29 , ERR_TELEMATICS_BT_NOT_AVAILABLE = 30 , ERR_TELEMATICS_GPS_NOT_AVAILABLE
= 31 ,
ERR_MEM_ALLOC_FAIL = 32 , ERR_JOIN_THREAD = 33 , ERR_INVALID_STARTUP_TRIGGER
= 34 , ERR_END }
• 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 ,
Analog_channel_END }
• enum videoStandard {
STD_M_J_NTSC = 0 , STD_B_D_G_H_I_N_PAL = 1 , STD_M_PAL = 2 , STD_PAL
= 3 ,
STD_NTSC = 4 , STD_SECAM = 5 }
• enum VideoRotation { RotNone = 0 , Rot90 , Rot180 , Rot270 }
• enum CanFrameType { FrameStandard , FrameExtended , FrameStandardExtended
}
• enum CanMode { CanModeClassic , CanModeFlexibleData }
• enum TriggerConf {
Front_Button_Enabled = 1 , OnOff_Signal_Enabled = 2 , Both_Button_And_Signal_Enabled
= 3 , CAN_Button_Activity = 5 ,
CAN_OnOff_Activity = 6 , CAN_Button_OnOff_Activity = 7 , CI_Button_Activity
= 9 , CI_OnOff_Activity = 10 ,
CI_Button_OnOff_Activity = 11 , CI_CAN_Button_Activity = 13 , CI_CAN_OnOff_Activity
= 14 , All_Events = 15 ,
Last_trigger_conf }
• 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 ,
BPTS_END }
• enum OCDStatus { OCD_OK = 0 , OCD_OC = 1 , OCD_POWER_OFF = 2 }
• 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 RS4XXPort { RS4XXPort1 = 1 , RS4XXPort2 , RS4XXPort3 , RS4XXPort4  
 }
- enum CfgInModeEnum {  
CFGIN\_NOT\_IN\_USE = 0 , CFGIN\_HI\_SWITCH , CFGIN\_LOW\_SWITCH ,  
CFGIN\_VOLTAGE\_2V5 ,  
CFGIN\_VOLTAGE\_5V , CFGIN\_RESISTANCE , CFGIN\_FREQ\_FLOATING ,  
CFGIN\_FREQ\_PULLUP ,  
CFGIN\_FREQ\_PULLDOWN , CFGIN\_RESISTANCE\_500 , CFGIN\_CURRENT\_4\_20  
 , CFGIN\_VOLTAGE\_10V ,  
CFGIN\_VOLTAGE\_32V , CFGIN\_DIGITAL\_PD\_5V , CFGIN\_DIGITAL\_PD\_10V  
 , CFGIN\_DIGITAL\_PD\_32V ,  
CFGIN\_DIGITAL\_F\_5V , CFGIN\_DIGITAL\_F\_10V , CFGIN\_DIGITAL\_F\_32V  
 , CFGIN\_DIGITAL\_PU\_5V ,  
CFGIN\_DIGITAL\_PU\_10V , CFGIN\_DIGITAL\_PU\_32V , CFGIN\_FREQ\_PD\_5V  
 , CFGIN\_FREQ\_PD\_10V ,  
CFGIN\_FREQ\_PD\_32V , CFGIN\_FREQ\_F\_5V , CFGIN\_FREQ\_F\_10V , CFGIN\_FREQ\_F\_32V  
 ,  
CFGIN\_FREQ\_PU\_5V , CFGIN\_FREQ\_PU\_10V , CFGIN\_FREQ\_PU\_32V , CFGIN\_VS\_FreqInMode1  
 ,  
CFGIN\_VS\_FreqInMode1PU , CFGIN\_VS\_FreqInMode2 , CFGIN\_VS\_FreqInMode2PU  
 , CFGIN\_MAX }
- enum ButtonConfigEnum {  
BUTTON\_ONLY\_MP\_ACTION = 0x00 , BUTTON\_AS\_STARTUP\_TRIG = 0x02  
 , BUTTON\_AS\_ACTION\_TRIG = 0x04 , BUTTON\_AS\_ACTION\_STARTUP\_TRIG  
 = 0x06 ,  
BUTTON\_AS\_BACKLIGHT\_DECREASE = 0x08 , BUTTON\_AS\_BACKLIGHT\_DECR\_STARTUP\_TRIG  
 = 0x0A , BUTTON\_AS\_BACKLIGHT\_INCREASE = 0x0C , BUTTON\_AS\_BACKLIGHT\_INCR\_STARTUP\_TRIG  
 = 0x0E }
- enum BootModeEnum {  
BOOTMODE\_EMMC = 0 , BOOTMODE\_SD , BOOTMODE\_SERIAL , BOOTMODE\_RESCUE  
 ,  
BOOTMODE\_RESCUE\_SPECIAL }
- enum ConfigOnOffTriggerMode { CONFIG\_ONOFF\_EDGE\_TRIGGER = 0 ,  
CONFIG\_ONOFF\_LEVEL\_TRIGGER }
- enum PowerOutput {  
PowerOutput1 = 0 , PowerOutput2 , PowerOutput3 , PowerOutput4 ,  
PowerOutput5 , PowerOutput6 , PowerOutputMax }
- enum SystemMode {  
SYSTEMMODE\_Startup = 0 , SYSTEMMODE\_StartupRescue = 1 , SYSTEMMODE\_StartupRescueFactoryRes  
 = 2 , SYSTEMMODE\_NormalRunning = 3 ,  
SYSTEMMODE\_RescueRunning = 4 , SYSTEMMODE\_RescueRunningFactoryReset  
 = 5 , SYSTEMMODE\_Unknown = 6 }



- enum `startupReasonCodes` {  
`startupReasonCodeUndefined` = 0x0000 , `startupReasonCodeButtonPress` = 0x0055  
`startupReasonCodeExtCtrl` = 0x00AA , `startupReasonCodeMPRestart` = 0x00F0 ,  
`startupReasonCodePowerOnStartup` = 0x000F , `startupReasonCodeCanActivity`  
= 0x003c , `startupReasonCodeCIActivity` = 0x00c3 , `startupReasonAlwaysStart`  
= 0x00e1 ,  
`startupReasonUnknownTrigger` = 0x001e }
- enum `shutDownReasonCodes` { `shutdownReasonCodeNoError` = 0x001F }
- enum `hwErrorStatusCodes` { `errCodeNoErr` = 0 }
- 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_CALIBRATION_WIDTH` = 9 , `TS_CALIBRATION_MEASUREMENTS`  
= 10 , `TS_RESTORE_DEFAULT_SETTINGS` = 11 ,  
`TS_TCHAUTOCAL` = 12 }
- 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 `ABOUTHANDLE` CCAUXDLL\_CALLING\_CONV `GetAbout` (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV `About_release` (`ABOUTHANDLE`)
- EXTERN\_C CCAUXDLL\_API `eErr` CCAUXDLL\_CALLING\_CONV `About_getMainPCBSerial` (`ABOUTHANDLE`, `char.t` \*buff, `int32.t` len)
- EXTERN\_C CCAUXDLL\_API `eErr` CCAUXDLL\_CALLING\_CONV `About_getUnitSerial` (`ABOUTHANDLE`, `char.t` \*buff, `int32.t` len)
- EXTERN\_C CCAUXDLL\_API `eErr` CCAUXDLL\_CALLING\_CONV `About_getMainPCBArt` (`ABOUTHANDLE`, `char.t` \*buff, `int32.t` length)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainManufacturingDate](#) (ABOUTHANDLE, char.t \*buff, int32.t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainHWversion](#) (ABOUTHANDLE, char.t \*buff, int32.t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainProdRev](#) (ABOUTHANDLE, char.t \*buff, int32.t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainProdArtNr](#) (ABOUTHANDLE, char.t \*buff, int32.t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfETHConnections](#) (ABOUTHANDLE, uint8.t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfCANConnections](#) (ABOUTHANDLE, uint8.t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfVideoConnections](#) (ABOUTHANDLE, uint8.t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfUSBConnections](#) (ABOUTHANDLE, uint8.t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfSerialConnections](#) (ABOUTHANDLE, uint8.t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfDigIOConnections](#) (ABOUTHANDLE, uint8.t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsDisplayAvailable](#) (ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsTouchScreenAvailable](#) (ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsPCBBluetoothAvailable](#) (ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getDisplayResolution](#) (ABOUTHANDLE, char.t \*buff, int32.t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnPCBSerial](#) (ABOUTHANDLE, char.t \*buff, int32.t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnPCBArt](#) (ABOUTHANDLE, char.t \*buff, int32.t length)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnManufacturingDate](#) (ABOUTHANDLE, char.t \*buff, int32.t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnHWversion](#) (ABOUTHANDLE, char.t \*buff, int32.t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsWLANMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsGPSMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsGPRSMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsBTMounted](#) (ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getFrontPcbRev](#) (ABOUTHANDLE, uint8.t \*major, uint8.t \*minor)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsIOExpanderMounted](#) (ABOUTHANDLE, bool \*mounted)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIOExpanderValue](#)  
(ABOUTHANDLE, uint16\_t \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_hasOsBooted](#)  
(ABOUTHANDLE, bool \*bootComplete)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsAnybusMounted](#)  
(ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfCfgInConnections](#)  
(ABOUTHANDLE, uint8\_t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfPWMOutConnections](#)  
(ABOUTHANDLE, uint8\_t \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfButtons](#)  
(ABOUTHANDLE, int32\_t \*numbuttons)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfAnalogInputs](#)  
(ABOUTHANDLE, int32\_t \*numanalogins)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getUserEepromData](#)  
(ABOUTHANDLE, char\_t \*buff, uint16\_t length)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_setUserEepromData](#)  
(ABOUTHANDLE, uint16\_t startpos, const char\_t \*buff, uint16\_t length)
- EXTERN\_C CCAUXDLL\_API ADCHANDLE CCAUXDLL\_CALLING\_CONV  
[GetAdc](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Adc\\_release](#)  
(ADCHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Adc\\_getVoltage](#)  
(ADCHANDLE, VoltageEnum selection, float64\_t \*value)
- EXTERN\_C CCAUXDLL\_API AUXVERSIONHANDLE CCAUXDLL\_CALLING\_CONV  
[GetAuxVersion](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [AuxVersion\\_release](#)  
(AUXVERSIONHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getFPGAVersion](#)  
(AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t  
\*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getSSVersion](#)  
(AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t  
\*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getFrontVersion](#)  
(AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t  
\*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getCCAuxVersion](#)  
(AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t  
\*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getOSVersion](#)  
(AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t  
\*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getCCAuxDrvVersion](#)  
(AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t  
\*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getCanFdMcuVersion](#)  
(AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t  
\*build)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getCanFdApiVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getFullOSVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build, char\_t \*tag, uint32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV AuxVersion\_getFullCCAuxVersion (AUXVERSIONHANDLE, uint8\_t \*major, uint8\_t \*minor, uint8\_t \*release, uint8\_t \*build, char\_t \*tag, uint32\_t len)
- EXTERN\_C CCAUXDLL\_API BACKLIGHTHANDLE CCAUXDLL\_CALLING\_CONV GetBacklight (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Backlight\_release (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getIntensity (BACKLIGHTHANDLE, uint8\_t \*intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_setIntensity (BACKLIGHTHANDLE, uint8\_t intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getStatus (BACKLIGHTHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getHWStatus (BACKLIGHTHANDLE, bool \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_startAutomaticBL (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_stopAutomaticBL (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getAutomaticBLStatus (BACKLIGHTHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_setAutomaticBLParams (BACKLIGHTHANDLE, bool bSoftTransitions)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getAutomaticBLParams (BACKLIGHTHANDLE, bool \*bSoftTransitions, float64\_t \*k)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_setAutomaticBLFilter (BACKLIGHTHANDLE, uint32\_t averageWndSize, uint32\_t rejectWndSize, uint32\_t rejectDeltaInLux, LightSensorSamplingMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getAutomaticBLFilter (BACKLIGHTHANDLE, uint32\_t \*averageWndSize, uint32\_t \*rejectWndSize, uint32\_t \*rejectDeltaInLux, LightSensorSamplingMode \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_getLedDimming (BACKLIGHTHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Backlight\_setLedDimming (BACKLIGHTHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API BATTERYHANDLE CCAUXDLL\_CALLING\_CONV GetBattery (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Battery\_release (BATTERYHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_isBatteryPresent (BATTERYHANDLE, bool \*batteryIsPresent)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getBatteryVoltageStatus (BATTERYHANDLE, uint8\_t \*batteryVoltagePercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getBatteryChargingStatus (BATTERYHANDLE, ChargingStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getPowerSource (BATTERYHANDLE, PowerSource \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getBatteryTemp (BATTERYHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getHwErrorStatus (BATTERYHANDLE, ErrorStatus \*errorCode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getTimer (BATTERYHANDLE, BatteryTimerType \*times)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getMinMaxTemp (BATTERYHANDLE, int16\_t \*minTemp, int16\_t \*maxTemp)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getBatteryHWversion (BATTERYHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getBatterySwVersion (BATTERYHANDLE, uint16\_t \*major, uint16\_t \*minor, uint16\_t \*release, uint16\_t \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Battery\_getBatterySerial (BATTERYHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API BUZZERHANDLE CCAUXDLL\_CALLING↔\_CONV GetBuzzer (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Buzzer\_release (BUZZERHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_getFrequency (BUZZERHANDLE, uint16\_t \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_getVolume (BUZZERHANDLE, uint16\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_getScaledVolume (BUZZERHANDLE, uint8\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_getTrigger (BUZZERHANDLE, bool \*trigger)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_setFrequency (BUZZERHANDLE, uint16\_t frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_setVolume (BUZZERHANDLE, uint16\_t volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_setScaledVolume (BUZZERHANDLE, uint8\_t volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_setTrigger (BUZZERHANDLE, bool trigger)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Buzzer\_buzze (BUZZERHANDLE, int32\_t time, bool blocking)
- EXTERN\_C CCAUXDLL\_API CANSETTINGHANDLE CCAUXDLL\_CALLING↔\_CONV GetCanSetting (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV CanSetting\_release (CANSETTINGHANDLE)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CanSetting\_getBaudrate (CANSETTINGHANDLE, uint8\_t net, uint16\_t \*baudrate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CanSetting\_getFrameType (CANSETTINGHANDLE, uint8\_t net, CanFrameType \*frameType)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CanSetting\_setBaudrate (CANSETTINGHANDLE, uint8\_t net, uint16\_t baudrate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CanSetting\_setFrameType (CANSETTINGHANDLE, uint8\_t net, CanFrameType frameType)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CanSetting\_getFlexibleDataBaudrates (CANSETTINGHANDLE, uint8\_t net, uint16\_t \*baudrateNominal, uint16\_t \*baudrate↔ Data)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CanSetting\_setFlexibleDataBaudrates (CANSETTINGHANDLE, uint8\_t net, uint16\_t baudrateNominal, uint16\_t baudrate↔ Data)
- EXTERN\_C CCAUXDLL\_API char\_t const \*CCAUXDLL\_CALLING\_CONV GetErrorStringA (eErr errCode)
- EXTERN\_C CCAUXDLL\_API wchar\_t const \*CCAUXDLL\_CALLING\_CONV GetErrorStringW (eErr errCode)
- EXTERN\_C CCAUXDLL\_API CFGINHANDLE CCAUXDLL\_CALLING\_↔ CONV GetCfgIn (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV CfgIn\_release (CFGINHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_setCfgInMode (CFGINHANDLE, uint8\_t channel, CfgInModeEnum set\_mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getCfgInMode (CFGINHANDLE, uint8\_t channel, CfgInModeEnum \*get\_mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getValue (CFGINHANDLE, uint8\_t channel, uint16\_t \*sample\_value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getPwmValue (CFGINHANDLE, uint8\_t channel, float32\_t \*frequency, uint8\_t \*duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getFrequencyValue (CFGINHANDLE, uint8\_t channel, float32\_t \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_getMinFrequencyThreshold (CFGINHANDLE, uint8\_t channel, float32\_t \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_setMinFrequencyThreshold (CFGINHANDLE, uint8\_t channel, float32\_t frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV CfgIn\_setFrequencyFilterLevel (CFGINHANDLE, uint8\_t level)
- EXTERN\_C CCAUXDLL\_API CONFIGHANDLE CCAUXDLL\_CALLING\_↔ CONV GetConfig ()
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Config\_release (CONFIGHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getStartupTriggerConfig (CONFIGHANDLE, TriggerConf \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getShortButtonPressAction (CONFIGHANDLE, PowerAction \*action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getLongButtonPressAction (CONFIGHANDLE, PowerAction \*action)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getOnOffSigAction (CONFIGHANDLE, PowerAction \*action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getFrontBtnTrigTime (CONFIGHANDLE, uint16\_t \*triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getExtOnOffSigTrigTime (CONFIGHANDLE, uint32\_t \*triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getButtonFunction (CONFIGHANDLE, uint8\_t button\_number, ButtonConfigEnum \*button\_config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getSuspendMaxTime (CONFIGHANDLE, uint16\_t \*maxTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getCanStartupPowerConfig (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getVideoStartupPowerConfig (CONFIGHANDLE, uint8\_t \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getExtFanStartupPowerConfig (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getStartupVoltageConfig (CONFIGHANDLE, float64\_t \*voltage)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getHeatingTempLimit (CONFIGHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getPowerOnStartup (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setStartupTriggerConfig (CONFIGHANDLE, TriggerConf conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setShortButtonPressAction (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setLongButtonPressAction (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setOnOffSigAction (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setFrontBtnTrigTime (CONFIGHANDLE, uint16\_t triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setExtOnOffSigTrigTime (CONFIGHANDLE, uint32\_t triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setButtonFunction (CONFIGHANDLE, uint8\_t button\_number, ButtonConfigEnum button\_config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setSuspendMaxTime (CONFIGHANDLE, uint16\_t maxTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setCanStartupPowerConfig (CONFIGHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setVideoStartupPowerConfig (CONFIGHANDLE, uint8\_t config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setExtFanStartupPowerConfig (CONFIGHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setStartupVoltageConfig (CONFIGHANDLE, float64\_t voltage)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setHeatingTempLimit (CONFIGHANDLE, int16\_t temperature)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setPowerOnStartup (CONFIGHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setRS485Enabled (CONFIGHANDLE, RS4XXPort port, bool enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getRS485Enabled (CONFIGHANDLE, RS4XXPort port, bool \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setOnOffTriggerMode (CONFIGHANDLE, ConfigOnOffTriggerMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getOnOffTriggerMode (CONFIGHANDLE, ConfigOnOffTriggerMode \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setNextBootMode (CONFIGHANDLE, BootModeEnum mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getNextBootMode (CONFIGHANDLE, BootModeEnum \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setOSAliveMonitoring (CONFIGHANDLE, CCStatus enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getOSAliveMonitoring (CONFIGHANDLE, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getOnOffSignalState (CONFIGHANDLE, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getDigPowerOutputStartupConfig (CONFIGHANDLE, PowerOutput output, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setDigPowerOutputStartupConfig (CONFIGHANDLE, PowerOutput output, CCStatus enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getShortBeepSettings (CONFIGHANDLE, uint16\_t \*duration, uint16\_t \*frequency, uint16\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setShortBeepSettings (CONFIGHANDLE, uint16\_t duration, uint16\_t frequency, uint16\_t volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_getLongBeepSettings (CONFIGHANDLE, uint16\_t \*duration, uint16\_t \*frequency, uint16\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Config\_setLongBeepSettings (CONFIGHANDLE, uint16\_t duration, uint16\_t frequency, uint16\_t volume)
- EXTERN\_C CCAUXDLL\_API DIAGNOSTICHANDLE CCAUXDLL\_CALLING\_CONV GetDiagnostic (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Diagnostic\_release (DIAGNOSTICHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getSSTemp (DIAGNOSTICHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPCBTemp (DIAGNOSTICHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPMTemp (DIAGNOSTICHANDLE, uint8\_t index, int16\_t \*temperature, JidaSensorType \*jst)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getStartupReason (DIAGNOSTICHANDLE, uint16\_t \*reason)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getShutDownReason (DIAGNOSTICHANDLE, uint16\_t \*reason)



- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getHwErrorStatus (DIAGNOSTICHANDLE, uint16\_t \*errorCode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getTimer (DIAGNOSTICHANDLE, TimerType \*times)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getMinMaxTemp (DIAGNOSTICHANDLE, int16\_t \*minTemp, int16\_t \*maxTemp)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPowerCycles (DIAGNOSTICHANDLE, uint16\_t \*powerCycles)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_clearHwErrorStatus (DIAGNOSTICHANDLE)
- EXTERN\_C CCAUXDLL\_API char\_t const \*CCAUXDLL\_CALLING\_CONV GetHwErrorStatusStringA (uint16\_t errCode)
- EXTERN\_C CCAUXDLL\_API wchar\_t const \*CCAUXDLL\_CALLING\_CONV GetHwErrorStatusStringW (uint16\_t errCode)
- EXTERN\_C CCAUXDLL\_API char\_t const \*CCAUXDLL\_CALLING\_CONV GetStartupReasonStringA (uint16\_t code)
- EXTERN\_C CCAUXDLL\_API wchar\_t const \*CCAUXDLL\_CALLING\_CONV GetStartupReasonStringW (uint16\_t code)
- EXTERN\_C CCAUXDLL\_API DIGIOHANDLE CCAUXDLL\_CALLING\_CONV GetDigIO (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV DigIO\_release (DIGIOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV DigIO\_getDigIO (DIGIOHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV DigIO\_setDigIO (DIGIOHANDLE, uint8\_t state)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV DigIO\_getDigPowerOutput (DIGIOHANDLE, PowerOutput output, CCStatus \*enabled, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV DigIO\_setDigPowerOutput (DIGIOHANDLE, PowerOutput output, CCStatus enabled)
- EXTERN\_C CCAUXDLL\_API FIRMWAREUPGHANDLE CCAUXDLL\_CALLING\_CONV GetFirmwareUpgrade (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_release (FIRMWAREUPGHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startFpgaUpgrade (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startFpgaVerification (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startSSUpgrade (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startSSVerification (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startFrontUpgrade (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startFrontVerification (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startDMCUUpgrade (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking, uint8\_t index)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_startDMCUVerification (FIRMWAREUPGHANDLE, const char\_t \*filename, bool blocking, uint8\_t index)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_getUpgradeStatus (FIRMWAREUPGHANDLE, UpgradeStatus \*status, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FirmwareUpgrade\_shutDown (FIRMWAREUPGHANDLE)
- EXTERN\_C CCAUXDLL\_API FRONTLEDHANDLE CCAUXDLL\_CALLING\_CONV GetFrontLED (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV FrontLED\_release (FRONTLEDHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getSignal (FRONTLEDHANDLE, float64\_t \*frequency, uint8\_t \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getOnTime (FRONTLEDHANDLE, uint8\_t \*onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getOffTime (FRONTLEDHANDLE, uint8\_t \*offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getIdleTime (FRONTLEDHANDLE, uint8\_t \*idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getNrOfPulses (FRONTLEDHANDLE, uint8\_t \*nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getColor (FRONTLEDHANDLE, uint8\_t \*red, uint8\_t \*green, uint8\_t \*blue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getStandardColor (FRONTLEDHANDLE, CCAuxColor \*color)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getEnabledDuringStartup (FRONTLEDHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setSignal (FRONTLEDHANDLE, float64\_t frequency, uint8\_t dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setOnTime (FRONTLEDHANDLE, uint8\_t onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setOffTime (FRONTLEDHANDLE, uint8\_t offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setIdleTime (FRONTLEDHANDLE, uint8\_t idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setNrOfPulses (FRONTLEDHANDLE, uint8\_t nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setColor (FRONTLEDHANDLE, uint8\_t red, uint8\_t green, uint8\_t blue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setStandardColor (FRONTLEDHANDLE, CCAuxColor color)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setOff (FRONTLEDHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setEnabledDuringStartup (FRONTLEDHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getBootLEDConfig (FRONTLEDHANDLE, uint8\_t \*red, uint8\_t \*green, uint8\_t \*blue, float32\_t \*frequency, uint8\_t \*dutyCycle)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setBootLEDConfig (FRONTLEDHANDLE, uint8\_t red, uint8\_t green, uint8\_t blue, float32\_t frequency, uint8\_t dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_getPostBootLEDConfig (FRONTLEDHANDLE, uint8\_t \*red, uint8\_t \*green, uint8\_t \*blue, float32\_t \*frequency, uint8\_t \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV FrontLED\_setPostBootLEDConfig (FRONTLEDHANDLE, uint8\_t red, uint8\_t green, uint8\_t blue, float32\_t frequency, uint8\_t dutyCycle)
- EXTERN\_C CCAUXDLL\_API LIGHTSENSORHANDLE CCAUXDLL\_CALLING\_CONV GetLightsensor (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Lightsensor\_release (LIGHTSENSORHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Lightsensor\_getIlluminance (LIGHTSENSORHANDLE, uint16\_t \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Lightsensor\_getIlluminance2 (LIGHTSENSORHANDLE, uint16\_t \*value, uint8\_t \*ch0, uint8\_t \*ch1)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Lightsensor\_getAverageIlluminance (LIGHTSENSORHANDLE, uint16\_t \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Lightsensor\_startAverageCalc (LIGHTSENSORHANDLE, uint32\_t averageWndSize, uint32\_t rejectWndSize, uint32\_t rejectDeltaInLux, LightSensorSamplingMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Lightsensor\_stopAverageCalc (LIGHTSENSORHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Lightsensor\_getOperatingRange (LIGHTSENSORHANDLE, LightSensorOperationRange \*range)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Lightsensor\_setOperatingRange (LIGHTSENSORHANDLE, LightSensorOperationRange range)
- EXTERN\_C CCAUXDLL\_API POWERHANDLE CCAUXDLL\_CALLING\_CONV GetPower (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Power\_release (POWERHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_getBLPowerStatus (POWERHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_getCanPowerStatus (POWERHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_getVideoPowerStatus (POWERHANDLE, uint8\_t \*videoStatus)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_getExtFanPowerStatus (POWERHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_getButtonPowerTransitionStatus (POWERHANDLE, ButtonPowerTransitionStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_getVideoOCDStatus (POWERHANDLE, OCDStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_getCanOCDStatus (POWERHANDLE, OCDStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_setBLPowerStatus (POWERHANDLE, CCStatus status)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_setCanPowerStatus (POWERHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_setVideoPowerStatus (POWERHANDLE, uint8\_t status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_setExtFanPowerStatus (POWERHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Power\_ackPowerRequest (POWERHANDLE)
- EXTERN\_C CCAUXDLL\_API POWERMGRHANDLE CCAUXDLL\_CALLING↔\_CONV GetPowerMgr (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV PowerMgr\_release (POWERMGRHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_registerControlledSuspendOrSH (POWERMGRHANDLE, PowerMgrConf conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_getConfiguration (POWERMGRHANDLE, PowerMgrConf \*conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_getPowerMgrStatus (POWERMGRHANDLE, PowerMgrStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_setAppReadyForSuspendOrSH (POWERMGRHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PowerMgr\_hasResumed (POWERMGRHANDLE, bool \*resumed)
- EXTERN\_C CCAUXDLL\_API PWMOUTHANDLE CCAUXDLL\_CALLING↔\_CONV GetPWMOut (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV PWMOut\_release (PWMOUTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_setPWMOutputChannelDutyC (PWMOUTHANDLE, uint8\_t channel, uint8\_t duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_setPWMOutputChannelFrequ (PWMOUTHANDLE, uint8\_t channel, float32\_t frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_getPWMOutputChannelDutyC (PWMOUTHANDLE, uint8\_t channel, uint8\_t \*duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_getPWMOutputChannelFrequ (PWMOUTHANDLE, uint8\_t channel, float32\_t \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_getPWMOutputStatus (PWMOUTHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV PWMOut\_setPWMOutOff (PWMOUTHANDLE, uint8\_t channel)
- EXTERN\_C CCAUXDLL\_API SMARTHANDLE CCAUXDLL\_CALLING↔\_CONV GetSmart (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Smart\_release (SMARTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getRemainingLifeTime (SMARTHANDLE, uint8\_t \*lifetimepercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getRemainingLifeTime2 (SMARTHANDLE, uint8\_t \*lifetimepercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getDeviceSerial (SMARTHANDLE, char\_t \*buff, int32\_t len)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getDeviceSerial2 (SMARTHANDLE, char\_t \*buff, int32\_t len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getInitialTime (SMARTHANDLE, time\_t \*time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Smart\_getInitialTime2 (SMARTHANDLE, time\_t \*time)
- EXTERN\_C CCAUXDLL\_API SOFTKEYHANDLE CCAUXDLL\_CALLING\_CONV GetSoftKey (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV SoftKey\_release (SOFTKEYHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getStatus (SOFTKEYHANDLE, uint16\_t \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightIntensity (SOFTKEYHANDLE, uint8\_t key, uint8\_t intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightIntensity (SOFTKEYHANDLE, uint8\_t key, uint8\_t \*intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setMultipleBacklightIntensities (SOFTKEYHANDLE, uint8\_t \*keys, uint8\_t \*intensities, uint8\_t array\_size)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getMultipleBacklightIntensities (SOFTKEYHANDLE, uint8\_t \*keys, uint8\_t \*intensities, uint8\_t array\_size)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightSignal (SOFTKEYHANDLE, float64\_t \*frequency, uint8\_t \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightOnTime (SOFTKEYHANDLE, uint8\_t \*onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightOffTime (SOFTKEYHANDLE, uint8\_t \*offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightIdleTime (SOFTKEYHANDLE, uint8\_t \*idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightNrOfPulses (SOFTKEYHANDLE, uint8\_t \*nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightSignal (SOFTKEYHANDLE, float64\_t frequency, uint8\_t dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightOnTime (SOFTKEYHANDLE, uint8\_t onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightOffTime (SOFTKEYHANDLE, uint8\_t offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightIdleTime (SOFTKEYHANDLE, uint8\_t idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightNrOfPulses (SOFTKEYHANDLE, uint8\_t nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightOff (SOFTKEYHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBacklightEnabledDuringStart (SOFTKEYHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBacklightEnabledDuringStart (SOFTKEYHANDLE, CCStatus status)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_getBootBacklightConfig (SOFTKEYHANDLE, uint8\_t \*bootIntensity, float32\_t \*bootFrequency, uint8\_t \*bootDutyCycle, uint8\_t \*postBootIntensity, float32\_t \*postBootFrequency, uint8\_t \*postBootDutyCycle, CCStatus \*postBootConfig)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV SoftKey\_setBootBacklightConfig (SOFTKEYHANDLE, uint8\_t bootIntensity, float32\_t bootFrequency, uint8\_t bootDutyCycle, uint8\_t postBootIntensity, float32\_t postBootFrequency, uint8\_t postBootDutyCycle, CCStatus postBootConfig)
- EXTERN\_C CCAUXDLL\_API TELEMATICSHANDLE CCAUXDLL\_CALLING\_CONV GetTelematics (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Telematics\_release (TELEMATICSHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getTelematicsAvailable (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPRSPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPRSStartupPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getWLANPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getWLANStartupPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getBTPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getBTStartupPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPSPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPSStartupPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPSAntennaStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setGPRSPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setGPRSStartupPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setWLANPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setWLANStartupPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setBTPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setBTStartupPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setGPSPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setGPSStartupPowerStatus (TELEMATICSHANDLE, CCStatus status)

- EXTERN\_C CCAUXDLL\_API TOUCHSCREENHANDLE CCAUXDLL\_CALLING\_↔  
\_CONV [GetTouchScreen](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [TouchScreen\\_release](#)  
(TOUCHSCREENHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getMode](#)  
(TOUCHSCREENHANDLE, [TouchScreenModeSettings](#) \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getMouseRightClickTime](#)  
(TOUCHSCREENHANDLE, uint16\_t \*time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setMode](#)  
(TOUCHSCREENHANDLE, [TouchScreenModeSettings](#) config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setMouseRightClickTime](#)  
(TOUCHSCREENHANDLE, uint16\_t time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setAdvancedSetting](#)  
(TOUCHSCREENHANDLE, [TSAdvancedSettingsParameter](#) param, uint16\_t data)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getAdvancedSetting](#)  
(TOUCHSCREENHANDLE, [TSAdvancedSettingsParameter](#) param, uint16\_t \*data)
- EXTERN\_C CCAUXDLL\_API TOUCHSCREENCALIBHANDLE CCAUXDLL\_↔  
\_CALLING\_CONV [GetTouchScreenCalib](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [TouchScreenCalib\\_release](#)  
(TOUCHSCREENCALIBHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreenCalib\\_setMode](#)  
(TOUCHSCREENCALIBHANDLE, [CalibrationModeSettings](#) mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreenCalib\\_getMode](#)  
(TOUCHSCREENCALIBHANDLE, [CalibrationModeSettings](#) \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreenCalib\\_setCalibrationPoint](#)  
(TOUCHSCREENCALIBHANDLE, uint8\_t pointNr)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreenCalib\\_checkCalibrationPoint](#)  
(TOUCHSCREENCALIBHANDLE, bool \*finished, uint8\_t pointNr)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreenCalib\\_getConfigParam](#)  
(TOUCHSCREENCALIBHANDLE, [CalibrationConfigParam](#) param, uint16\_t  
\*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreenCalib\\_setConfigParam](#)  
(TOUCHSCREENCALIBHANDLE, [CalibrationConfigParam](#) param, uint16\_t  
value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreenCalib\\_autoSensorCalib](#)  
(TOUCHSCREENCALIBHANDLE)
- EXTERN\_C CCAUXDLL\_API VIDEOHANDLE CCAUXDLL\_CALLING\_↔  
CONV [GetVideo](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Video\\_release](#)  
(VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_init](#)  
(VIDEOHANDLE, uint8\_t deviceNr)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_showVideo](#)  
(VIDEOHANDLE, bool show)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_setDeInterlaceMode](#)  
(VIDEOHANDLE, [DeInterlaceMode](#) mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_getDeInterlaceMode](#)  
(VIDEOHANDLE, [DeInterlaceMode](#) \*mode)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setMirroring (VIDEOHANDLE, CCStatus mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getMirroring (VIDEOHANDLE, CCStatus \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setRotation (VIDEOHANDLE, VideoRotation rotation)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getRotation (VIDEOHANDLE, VideoRotation \*rotation)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setActiveChannel (VIDEOHANDLE, VideoChannel channel)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getActiveChannel (VIDEOHANDLE, VideoChannel \*channel)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setColorKeys (VIDEOHANDLE, uint8\_t rKey, uint8\_t gKey, uint8\_t bKey)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getColorKeys (VIDEOHANDLE, uint8\_t \*rKey, uint8\_t \*gKey, uint8\_t \*bKey)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setVideoArea (VIDEOHANDLE, uint16\_t topLeftX, uint16\_t topLeftY, uint16\_t bottomRightX, uint16\_t bottomRightY)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getRawImage (VIDEOHANDLE, uint16\_t \*width, uint16\_t \*height, float32\_t \*frameRate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getVideoArea (VIDEOHANDLE, uint16\_t \*topLeftX, uint16\_t \*topLeftY, uint16\_t \*bottomRightX, uint16\_t \*bottomRightY)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getVideoStandard (VIDEOHANDLE, videoStandard \*standard)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getStatus (VIDEOHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setScaling (VIDEOHANDLE, float32\_t x, float32\_t y)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getScaling (VIDEOHANDLE, float32\_t \*x, float32\_t \*y)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_activateSnapshot (VIDEOHANDLE, bool activate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_takeSnapshot (VIDEOHANDLE, const char\_t \*path, bool bInterlaced)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_takeSnapshotRaw (VIDEOHANDLE, char\_t \*rawImgBuffer, uint32\_t rawImgBufSize, bool bInterlaced)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_takeSnapshotBmp (VIDEOHANDLE, char\_t \*\*bmpBuffer, uint32\_t \*bmpBufSize, bool bInterlaced, bool bNTSCFormat)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_createBitmap (VIDEOHANDLE, char\_t \*\*bmpBuffer, uint32\_t \*bmpBufSize, const char\_t \*rawImgBuffer, uint32\_t rawImgBufSize, bool bInterlaced, bool bNTSCFormat)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_freeBmpBuffer (VIDEOHANDLE, char\_t \*bmpBuffer)



- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_minimize (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_restore (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setDecoderReg (VIDEOHANDLE, uint8\_t decoderRegister, uint8\_t registerValue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getDecoderReg (VIDEOHANDLE, uint8\_t decoderRegister, uint8\_t \*registerValue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setCropping (VIDEOHANDLE, uint8\_t top, uint8\_t left, uint8\_t bottom, uint8\_t right)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getCropping (VIDEOHANDLE, uint8\_t \*top, uint8\_t \*left, uint8\_t \*bottom, uint8\_t \*right)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_showFrame (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setGraphicsOverlay (VIDEOHANDLE, CCStatus mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getGraphicsOverlay (VIDEOHANDLE, CCStatus \*mode)

## Variables

- const uint8\_t Video1Conf = (1 << 0)
- const uint8\_t Video2Conf = (1 << 1)
- const uint8\_t Video3Conf = (1 << 2)
- const uint8\_t Video4Conf = (1 << 3)
- const uint8\_t DigitalIn\_1 = (1 << 0)
- const uint8\_t DigitalIn\_2 = (1 << 1)
- const uint8\_t DigitalIn\_3 = (1 << 2)
- const uint8\_t DigitalIn\_4 = (1 << 3)

## 20.1.1 Typedef Documentation

### ABOUTHANDLE

```
typedef void* ABOUTHANDLE
```

### ADCHANDLE

```
typedef void* ADCHANDLE
```

### AUXVERSIONHANDLE

```
typedef void* AUXVERSIONHANDLE
```

**BACKLIGHTHANDLE**

```
typedef void* BACKLIGHTHANDLE
```

**BATTERYHANDLE**

```
typedef void* BATTERYHANDLE
```

**BUZZERHANDLE**

```
typedef void* BUZZERHANDLE
```

**CANSETTINGHANDLE**

```
typedef void* CANSETTINGHANDLE
```

**CFGINHANDLE**

```
typedef void* CFGINHANDLE
```

**CONFIGHANDLE**

```
typedef void* CONFIGHANDLE
```

**DIAGNOSTICHANDLE**

```
typedef void* DIAGNOSTICHANDLE
```

**DIGIOHANDLE**

```
typedef void* DIGIOHANDLE
```

**FIRMWAREUPGHANDLE**

```
typedef void* FIRMWAREUPGHANDLE
```

**FRONTLEDHANDLE**

```
typedef void* FRONTLEDHANDLE
```

**LIGHTSENSORHANDLE**

```
typedef void* LIGHTSENSORHANDLE
```

**POWERHANDLE**

```
typedef void* POWERHANDLE
```

**POWERMGRHANDLE**

```
typedef void* POWERMGRHANDLE
```

**PWMOUTHANDLE**

```
typedef void* PWMOUTHANDLE
```

**SMARTHANDLE**

```
typedef void* SMARTHANDLE
```

**SOFTKEYHANDLE**

```
typedef void* SOFTKEYHANDLE
```

**TELEMATICSHANDLE**

```
typedef void* TELEMATICSHANDLE
```

**TOUCHSCREENCALIBHANDLE**

```
typedef void* TOUCHSCREENCALIBHANDLE
```

**TOUCHSCREENHANDLE**

```
typedef void* TOUCHSCREENHANDLE
```

**VersionType**

```
typedef struct version_info VersionType
```

**VIDEOHANDLE**

```
typedef void* VIDEOHANDLE
```

## 20.1.2 Enumeration Type Documentation

### BootModeEnum

enum [BootModeEnum](#)

Enumeration of Bootmodes

Enumerator

BOOTMODE.EMMC	
BOOTMODE.SD	Boot from eMMC
BOOTMODE.SERIAL	Boot from SD
BOOTMODE.RESCUE	Serial download mode
BOOTMODE.RESCUE.SPECIAL	Boot to rescue system Boot to rescue system. Platform depended special.

### ButtonConfigEnum

enum [ButtonConfigEnum](#)

Enumeration of Button Configuration (bitfield representation)

Enumerator

BUTTON_ONLY_MP_ACTION	
BUTTON_AS_STARTUP_TRIG	Buttons are only read by Main Processor, i.e. normal button action which is handled in application space
BUTTON_AS_ACTION_TRIG	Set button to be used as startup trigger, in addition to MP application event
BUTTON_AS_ACTION_STARTUP_↔ TRIG	Set button to trigger suspend, shutdown or hard shutdown actions
BUTTON_AS_BACKLIGHT_↔ DECREASE	Set button to trigger startup, suspend, shutdown or hard shutdown actions
BUTTON_AS_BACKLIGHT_DECR_↔ _STARTUP_TRIG	Set button to act as backlight decrease button
BUTTON_AS_BACKLIGHT_↔ INCREASE	Set button to act as backlight decrease and startup trigger
BUTTON_AS_BACKLIGHT_INCR_↔ STARTUP_TRIG	Set button to act as backlight increase button

**ButtonPowerTransitionStatus**enum `ButtonPowerTransitionStatus`

Current status for front panel button and on/off signal. If any of them generate a suspend or shutdown event, it may also be read, briefly. When the button/signal is released, typically `BPTS_Suspend` or `BPTS_ShutDown` follows. Note: Do not rely on getting `BPTS_ShutDown` or `BPTS_Suspend` from user applications. The system is likely to start shutting down before the status can be read. Instead, use the `PowerMgr` class for handling system shutdown/suspend events.

Enumerator

<code>BPTS_No_Change</code>	No change
<code>BPTS_ShutDown</code>	A shutdown has been initiated since the front panel button has been pressed longer than the set <code>FrontBtnShutDownTrigTime</code>
<code>BPTS_Suspend</code>	Suspend mode has been initiated since the front panel button has been pressed (shortly) and suspend mode is enabled
<code>BPTS_Restart</code>	Not currently in use
<code>BPTS_BtnPressed</code>	The front panel button is currently pressed. It has not been released and it has not yet been held longer than <code>FrontBtnShutDownTrigTime</code> .
<code>BPTS_BtnPressedLong</code>	The front panel button is currently pressed. It has not been released and it has been held longer than <code>FrontBtnShutDownTrigTime</code> .
<code>BPTS_SignalOff</code>	The external on/off signal is low, but not yet long enough for the <code>ExtOnOffSigSuspTrigTime</code> .
<code>BPTS_END</code>	Mark end of BPTS - list

**CalibrationConfigParam**enum `CalibrationConfigParam`

Touch screen calibration parameters

Enumerator

<code>CONFIG_CALIBRATION_WITH</code>	
<code>CONFIG_CALIBRATION_↔ MEASUREMENTS</code>	Accepted error value when calibrating.
<code>CONFIG_5P_CALIBRATION_↔ POINT_BORDER</code>	Number of measurements to accept a calibration point.
<code>CONFIG_13P_CALIBRATION_↔ POINT_BORDER</code>	The number of pixels from the border where the 5 point calibration points should be located.

Enumerator

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.

### CalibrationModeSettings

enum `CalibrationModeSettings`  
Touch screen calibration modes

Enumerator

MODE_UNKNOWN	
MODE_NORMAL	Unknown mode.
MODE_CALIBRATION_5P	Normal operation mode.
MODE_CALIBRATION_9P	Calibration with 5 points mode.
MODE_CALIBRATION_13P	Calibration with 9 points mode.

### CanFrameType

enum `CanFrameType`  
Can frame type settings

Enumerator

FrameStandard	
FrameExtended	
FrameStandardExtended	

### CanMode

enum `CanMode`  
Can mode settings

Enumerator

CanModeClassic	
CanModeFlexibleData	

**CCAuxColor**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

**CCStatus**enum `CCStatus`

Enable/disable enumeration

Enumerator

Disabled	
Enabled	The setting is disabled or turned off

**CfgInModeEnum**enum `CfgInModeEnum`

Enumeration of ConfigurableInput modes

Enumerator

CFGIN_NOT_IN_USE	
CFGIN_HI_SWITCH	Disable configurable input measurement.
CFGIN_LOW_SWITCH	Read digital input value with <code>CfgIn.getValue</code>
CFGIN_VOLTAGE_2V5	Read digital input value with <code>CfgIn.getValue</code>
CFGIN_VOLTAGE_5V	Read voltage input value with <code>CfgIn.getValue</code> , 2.5V range
CFGIN_RESISTANCE	Read voltage input value with <code>CfgIn.getValue</code> , 5V range

## Enumerator

CFGIN_FREQ_FLOATING	Read resistance input value with CfgIn_getValue, ports 1-4 only
CFGIN_FREQ_PULLUP	Read frequency value with CfgIn_getPwmValue
CFGIN_FREQ_PULLDOWN	Read frequency value with CfgIn_getPwmValue
CFGIN_RESISTANCE_500	Read frequency value with CfgIn_getPwmValue
CFGIN_CURRENT_4_20	Read resistance input value with CfgIn_getValue, 0-500Ohm range, VA only, ports 1-4 only
CFGIN_VOLTAGE_10V	Read current input value with CfgIn_getValuerange 4-20 mA, VA only.
CFGIN_VOLTAGE_32V	Read voltage input value with CfgIn_getValue, 10V range, VA only
CFGIN_DIGITAL_PD_5V	Read voltage input value with CfgIn_getValue, 32V range, VA only
CFGIN_DIGITAL_PD_10V	Read digital input value with CfgIn_getValue, pull-down, 5V range, 2.5V threshold, VA only
CFGIN_DIGITAL_PD_32V	Read digital input value with CfgIn_getValue, pull-down, 10V range, 5V threshold, VA only
CFGIN_DIGITAL_F_5V	Read digital input value with CfgIn_getValue, pull-down, 32V range, 10V threshold, VA only
CFGIN_DIGITAL_F_10V	Read digital input value with CfgIn_getValue, floating, 5V range, 2.5V threshold, VA only
CFGIN_DIGITAL_F_32V	Read digital input value with CfgIn_getValue, floating, 10V range, 5V threshold, VA only
CFGIN_DIGITAL_PU_5V	Read digital input value with CfgIn_getValue, floating, 32V range, 10V threshold, VA only
CFGIN_DIGITAL_PU_10V	Read digital input value with CfgIn_getValue, pull-up, 5V range, 2.5V threshold, VA only, ports 5-8 only
CFGIN_DIGITAL_PU_32V	Read digital input value with CfgIn_getValue, pull-up, 10V range, 5V threshold, VA only, ports 5-8 only
CFGIN_FREQ_PD_5V	Read digital input value with CfgIn_getValue, pull-up, 32V range, 10V threshold, VA only, ports 5-8 only
CFGIN_FREQ_PD_10V	Read frequency value with CfgIn_getFrequencyValue, pull-down, 5V range, 2.5V threshold, VA only
CFGIN_FREQ_PD_32V	Read frequency value with CfgIn_getFrequencyValue, pull-down, 10V range, 5V threshold, VA only
CFGIN_FREQ_F_5V	Read frequency value with CfgIn_getFrequencyValue, pull-down, 32V range, 10V threshold, VA only



## Enumerator

CFGIN_FREQ_F_10V	Read frequency value with CfgIn_getFrequencyValue, floating, 5V range, 2.5V threshold, VA only
CFGIN_FREQ_F_32V	Read frequency value with CfgIn_getFrequencyValue, floating, 10V range, 5V threshold, VA only
CFGIN_FREQ_PU_5V	Read frequency value with CfgIn_getFrequencyValue, floating, 32V range, 10V threshold, VA only
CFGIN_FREQ_PU_10V	Read frequency value with CfgIn_getFrequencyValue, pull-up, 5V range, 2.5V threshold, VA only, ports 5-8 only
CFGIN_FREQ_PU_32V	Read frequency value with CfgIn_getFrequencyValue, pull-up, 10V range, 5V threshold, VA only, ports 5-8 only
CFGIN_VS_FreqInMode1	Read frequency value with CfgIn_getFrequencyValue, pull-up, 32V range, 10V threshold, VA only, ports 5-8 only
CFGIN_VS_FreqInMode1PU	Mode 1, High level 3420mV, Low level 2540mV, VS only
CFGIN_VS_FreqInMode2	Mode 1, High level 3420mV, Low level 2540mV, with pull-up, VS only
CFGIN_VS_FreqInMode2PU	Mode 2, High level 6280mV, Low level 1520mV, VS only
CFGIN_MAX	Mode 2, High level 6280mV, Low level 1520mV, with pull-up, VS only

**ChargingStatus**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

Enumerator

ChargingStatus_Unknown	There was an error determining the charging status
------------------------	--

### ConfigOnOffTriggerMode

enum `ConfigOnOffTriggerMode`

Enumeration of OnOff/Ignition/KeySwitch signal trigger modes

Enumerator

CONFIG_ONOFF_EDGE_TRIGGER	
CONFIG_ONOFF_LEVEL_TRIGGER	OnOff/Ignition/KeySwitch signal trigger on signal edge (default) OnOff/Ignition/KeySwitch signal trigger on signal level

### DeInterlaceMode

enum `DeInterlaceMode`

The available deinterlace modes

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

### eErr

enum `eErr`

Error code enumeration

Enumerator

ERR_SUCCESS	
ERR_OPEN_FAILED	Success
ERR_NOT_SUPPORTED	Open failed
ERR_UNKNOWN_FEATURE	Not supported

## Enumerator

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

Enumerator

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
ERR_JOIN_THREAD	Failed to allocate memory
ERR_INVALID_STARTUP_TRIGGER	Failed to join thread
ERR_END	Start-up trigger or button configuration set to an invalid value, see enum TriggerConf for more details. Marks end of errors - list

### ErrorStatus

enum `ErrorStatus`

Error status.

Enumerator

ErrorStatus_NoError	
ErrorStatus_ThermistorTempSensor	
ErrorStatus_SecondaryTempSensor	
ErrorStatus_ChargeFail	
ErrorStatus_Overcurrent	
ErrorStatus_Init	

### hwErrorStatusCodes

enum `hwErrorStatusCodes`

The error codes returned by `getHWErrorStatus`.

Enumerator

errCodeNoErr	
--------------	--

**JidaSensorType**enum `JidaSensorType`

Jida temperature sensor types

Enumerator

TEMP_CPU	
TEMP_BOX	
TEMP_ENV	
TEMP_BOARD	
TEMP_BACKPLANE	
TEMP_CHIPSETS	
TEMP_VIDEO	
TEMP_OTHER	

**LightSensorOperationRange**enum `LightSensorOperationRange`

Light sensor operation ranges.

Enumerator

RangeStandard	
RangeExtended	Light sensor operation range standard

**LightSensorSamplingMode**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.

**OCDStatus**enum `OCDStatus`

Overcurrent detection status.

Enumerator

<code>OCD_OK</code>	Normal operation, no overcurrent condition detected
<code>OCD_OC</code>	Overcurrent has been detected, power has therefore been turned off, but may be functioning again if the problem disappeared after re-test
<code>OCD_POWER_OFF</code>	Overcurrent has been detected, power has been permanently turned off

**PowerAction**enum `PowerAction`

Button and on/off signal actions.

Enumerator

<code>NoAction</code>	No action taken
<code>ActionSuspend</code>	The system enters suspend mode
<code>ActionShutDown</code>	The system shuts down

**PowerMgrConf**enum `PowerMgrConf`

Enumeration of the settings that can be used with the PowerMgr system.

Enumerator

<code>Normal</code>	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.
<code>ApplicationControlled</code>	Applications can delay suspend/shutdown requests.
<code>BatterySuspend</code>	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 <code>ApplicationControlled</code> .

**PowerMgrStatus**enum `PowerMgrStatus`

Enumerator

NoRequestsPending	No suspend or shutdown requests.
SuspendPending	A suspend request is pending.
ShutdownPending	A shutdown request is pending.

**PowerOutput**enum `PowerOutput`

Enumerator

PowerOutput1	
PowerOutput2	
PowerOutput3	
PowerOutput4	
PowerOutput5	
PowerOutput6	
PowerOutputMax	

**PowerSource**enum `PowerSource`

Current power source of the computer.

Enumerator

PowerSource_Battery	
PowerSource_ExternalPower	

**RS4XXPort**enum `RS4XXPort`

Enumeration of RS4XX ports (1-4)

Enumerator

RS4XXPort1	
RS4XXPort2	
RS4XXPort3	
RS4XXPort4	

### shutDownReasonCodes

enum `shutDownReasonCodes`

The shutdown codes returned by `getShutDownReason`.

Enumerator

shutdownReasonCodeNoError	
---------------------------	--

### startupReasonCodes

enum `startupReasonCodes`

The restart codes returned by `getStartupReason`.

Enumerator

<code>startupReasonCodeUndefined</code>	
<code>startupReasonCodeButtonPress</code>	Unknown startup reason.
<code>startupReasonCodeExtCtrl</code>	The system was started by front panel button press
<code>startupReasonCodeMPRestart</code>	The system was started by the external control signal
<code>startupReasonCodePowerOnStartup</code>	The system was restarted by OS request
<code>startupReasonCodeCanActivity</code>	The system was started due to the PowerOnStartup setting
<code>startupReasonCodeCIActivity</code>	The system was started due to activity on the Can bus (CCpilot VC family)
<code>startupReasonAlwaysStart</code>	The system was started due to activity on the configurable input signals (CCpilot VC family)
<code>startupReasonUnknownTrigger</code>	The system was prevented to shutdown, since it is not allowed on this unit type.



**SystemMode**enum `SystemMode`

Enumeration of system running modes

Enumerator

SYSTEMMODE.Startup	
SYSTEMMODE.StartupRescue	
SYSTEMMODE.StartupRescueFactoryReset	
SYSTEMMODE.NormalRunning	
SYSTEMMODE.RescueRunning	
SYSTEMMODE.RescueRunningFactoryReset	
SYSTEMMODE.Unknown	

**TouchScreenModeSettings**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.

**TriggerConf**enum `TriggerConf`

Trigger configuration enumeration. Valid settings for enabling of front button and external on/off signal. For platforms XM, XL and XA platforms, front button and on/off (ignition) signal can be configured.

For the VC platform, CI state activity and Can data reception can also be used as wakeup sources from suspend mode. bit 0 - enable wakeup by front button (from OFF and suspend mode) bit 1 - enable wakeup by on/off (ignition) signal (from OFF and suspend mode) bit 2 - enable wakeup by CAN activity (from suspend mode, VC only) bit 3 - enable wakeup by CI (Configurable input) state change (from suspend mode, VC only)

Note that there must always be a way to start the unit from shutdown mode. Therefore, at least one of the following must be true:

- Front button enabled as start-up trigger AND (CCpilot VC) at least one button configured as start-up trigger
- External on/off (ignition) signal configured as start-up trigger.

## 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
CAN_Button_Activity	VC platform, wake up on CAN and Buttons
CAN_OnOff_Activity	VC platform, wake up on CAN and On/Off/Ignition signal
CAN_Button_OnOff_Activity	VC platform, wake up on CAN, Buttons and On/Off/Ignition signal
CI_Button_Activity	VC platform, wake up on CI and Button State Change
CI_OnOff_Activity	VC platform, wake up on CI and On/Off/Ignition signal State Change
CI_Button_OnOff_Activity	VC platform, wake up on CI, Button and On/Off/Ignition signal State Change
CI.CAN_Button_Activity	VC platform, wake up on CI, CAN and Button State Change
CI.CAN_OnOff_Activity	VC platform, wake up on CI, CAN and On/Off/Ignition signal State Change
All_Events	VC platform, wake up on all events
Last_trigger_conf	

**TSAdvancedSettingsParameter**enum [TSAdvancedSettingsParameter](#)

Touch screen advanced settings parameters

## Enumerator

TS.RIGHT_CLICK_TIME	Right click time in ms, except for touch profile on XM platform
TS.LOW_LEVEL	Lowest A/D value required for registering a touch event. Front uc 0.5.3.1 had the default value of 3300, newer versions: 3400.

## Enumerator

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 had 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 had 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_CALIBRATION_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.

Enumerator

TS_TCHAUTOCAL	Time (in units of 200 ms) until the touch screen is recalibrated when continuously touching the screen at one point. A setting of zero disables the recalibration. Valid for PCAP touch panels only. Device must be restarted for changes to have any effect. The default value is 50 which corresponds to 10 seconds.
---------------	--

### UpgradeAction

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

### VideoChannel

enum `VideoChannel`

The available analog video channels

Enumerator

Analog_Channel_1	
Analog_Channel_2	
Analog_Channel_3	
Analog_Channel_4	
Analog_channel_END	

**VideoRotation**enum `VideoRotation`

Enumerator

RotNone	
Rot90	
Rot180	
Rot270	

**videoStandard**enum `videoStandard`

Enumerator

STD_M_J_NTSC	
STD_B_D_G_H_I_N_PAL	(M,J) NTSC ITU-R BT.601
STD_M_PAL	(B, D, G, H, I, N) PAL ITU-R BT.601
STD_PAL	(M) PAL ITU-R BT.601
STD_NTSC	PAL-Nc ITU-R BT.601
STD_SECAM	NTSC 4.43 ITU-R BT.601

**VoltageEnum**enum `VoltageEnum`

Voltage type enumeration

Enumerator

VOLTAGE_24VIN	
VOLTAGE_24V	< 24VIN
VOLTAGE_12V	< 24V
VOLTAGE_12VID	< 12V
VOLTAGE_5V	< 12VID
VOLTAGE_3V3	< 5V

## Enumerator

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
VOLTAGE_24V_BACKUP	< SS internal VRef
VOLTAGE_2V5	< 24V backup capacitor
VOLTAGE_1V1	< 2.5V
VOLTAGE_1V3_PER	< 1.1V
VOLTAGE_1V3_VDDA	< 1.3V_PER
VOLTAGE_3V3STBY	< 1.3V_VDDA
VOLTAGE_VPMIC	< 3.3V STBY VC
VOLTAGE_VMAIN	< V PMIC VC
VOLTAGE_UB	< V MAIN VC
VOLTAGE_AI.1	< V UB VI2
VOLTAGE_AI.2	< V AI.1 VI2
VOLTAGE_AI.3	< V AI.2 VI2
VOLTAGE_5V_MP	< V AI.3 VI2
VOLTAGE_12V_BL	< 5V_MP

Enumerator

VOLTAGE_VHEAT	< 12V_BL
VOLTAGE_VIN_FILT	< VHEAT
VOLTAGE_VIN_PROT	< VIN_FILT
VOLTAGE_BL_ANODE	< VIN_PROT
VOLTAGE_IGN_STATE	< BL_ANODE
VOLTAGE_IGN_VALUE	< IGN_STATUS
VOLTAGE_PRE_IGN_STATE	< IGN_VALUE
VOLTAGE_PRE_IGN_VALUE	< PRE_IGN_STATUS
VOLTAGE_VBACKUP	< PRE_IGN_VALUE
VOLTAGE_END	< VBACKUP < Marks end of voltages - list

### 20.1.3 Function Documentation

#### GetErrorStringA()

```
EXTERN_C CCAUXDLL_API char_t const* CCAUXDLL_CALLING_CONV CrossControl::Get↔
ErrorStringA (
    eErr errCode )
```

to get a string description.

Get a string description of an error code.

Supported Platform(s): XL, XL5, XM, XS, XA, VS, VI2

Parameters

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

**Returns**

String description of an error code.

**GetErrorStringW()**

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

Get a string description of an error code.

Supported Platform(s): XL, XL5, XM, XS, XA, VS, VI2

**Parameters**

<i>errCode</i>	An error code for which
----------------	-------------------------

**Returns**

String description of an error code.

**GetHwErrorStatusStringA()**

```
EXTERN_C CCAUXDLL_API char_t const* CCAUXDLL_CALLING_CONV CrossControl::GetHwErrorStatusStringA (
    uint16_t errCode )
```

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

Supported Platform(s): All

**Parameters**

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

**Returns**

String description of an error code.

**GetHwErrorStatusStringW()**

```
EXTERN_C CCAUXDLL_API wchar_t const* CCAUXDLL_CALLING_CONV CrossControl::GetHwErrorStatusStringW (
    uint16_t errCode )
```

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

Supported Platform(s): All



## Parameters

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

## Returns

String description of an error code.

**GetStartupReasonStringA()**

```
EXTERN_C CCAUXDLLAPI char_t const* CCAUXDLL_CALLING_CONV CrossControl::GetStartupReasonStringA (
    uint16_t code )
```

Get a string description of a startup reason code returned from `getStartupReason`.  
Supported Platform(s): All

## Parameters

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

## Returns

String description of a code.

**GetStartupReasonStringW()**

```
EXTERN_C CCAUXDLLAPI wchar_t const* CCAUXDLL_CALLING_CONV CrossControl::GetStartupReasonStringW (
    uint16_t code )
```

Get a string description of a startup reason code returned from `getStartupReason`.  
Supported Platform(s): All

## Parameters

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

## Returns

String description of a code.

**20.1.4 Variable Documentation**

**DigitalIn\_1**

```
const uint8_t DigitalIn_1 = (1 << 0)
    Bit defines for getDigIO
```

**DigitalIn\_2**

```
const uint8_t DigitalIn_2 = (1 << 1)
```

**DigitalIn\_3**

```
const uint8_t DigitalIn_3 = (1 << 2)
```

**DigitalIn\_4**

```
const uint8_t DigitalIn_4 = (1 << 3)
```

**Video1Conf**

```
const uint8_t Video1Conf = (1 << 0)
    Bit defines for getVideoStartupPowerConfig and setVideoStartupPowerConfig
```

**Video2Conf**

```
const uint8_t Video2Conf = (1 << 1)
    Video channel 1 config
```

**Video3Conf**

```
const uint8_t Video3Conf = (1 << 2)
    Video channel 2 config
```

**Video4Conf**

```
const uint8_t Video4Conf = (1 << 3)
    Video channel 3 config
```

# Chapter 21

## Data Structure Documentation

### 21.1 BatteryTimerType Struct Reference

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

#### Data Fields

- [uint32\\_t TotRunTimeMain](#)
- [uint32\\_t TotRunTimeBattery](#)
- [uint32\\_t RunTime\\_m20](#)
- [uint32\\_t RunTime\\_m20\\_0](#)
- [uint32\\_t RunTime\\_0\\_40](#)
- [uint32\\_t RunTime\\_40\\_60](#)
- [uint32\\_t RunTime\\_60\\_70](#)
- [uint32\\_t RunTime\\_70\\_80](#)
- [uint32\\_t RunTime\\_Above80](#)

#### 21.1.1 Field Documentation

##### RunTime\_0\_40

[uint32\\_t](#) RunTime\_0\_40

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

##### RunTime\_40\_60

[uint32\\_t](#) RunTime\_40\_60

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

##### RunTime\_60\_70

[uint32\\_t](#) RunTime\_60\_70

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

**RunTime\_70\_80**`uint32_t` RunTime\_70\_80

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

**RunTime\_Above80**`uint32_t` RunTime\_Above80

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

**RunTime\_m20**`uint32_t` RunTime\_m20

Total running time on battery power (minutes)

**RunTime\_m20\_0**`uint32_t` RunTime\_m20\_0

Total runtime below -20 deg C (minutes)

**TotRunTimeBattery**`uint32_t` TotRunTimeBattery

Total running time on main power (minutes)

**TotRunTimeMain**`uint32_t` TotRunTimeMain

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

- IncludeFiles/[Battery.h](#)

## 21.2 BuzzerSetup Struct Reference

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

### Data Fields

- `uint16_t` frequency
- `uint16_t` volume

#### 21.2.1 Field Documentation

**frequency**`uint16_t` frequency

buzzer frequency

**volume**

```
uint16_t volume
```

buzzer volume

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

- [IncludeFiles/CCAuxTypes.h](#)

## 21.3 FpgaLedTimingType Struct Reference

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

### Data Fields

- [uint8\\_t ledNbr](#)
- [uint8\\_t onTime](#)
- [uint8\\_t offTime](#)
- [uint8\\_t idleTime](#)
- [uint8\\_t nrOfPulses](#)

### 21.3.1 Field Documentation

**idleTime**

```
uint8_t idleTime
```

LED idle time in 100ms

**ledNbr**

```
uint8_t ledNbr
```

Number of LED

**nrOfPulses**

```
uint8_t nrOfPulses
```

Pulses per sequences

**offTime**

```
uint8_t offTime
```

LED off time in 10ms

**onTime**

```
uint8_t onTime
```

LED on time in 10ms

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

- [IncludeFiles/CCAuxTypes.h](#)

## 21.4 LedColorMixType Struct Reference

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

### Data Fields

- [uint8\\_t red](#)
- [uint8\\_t green](#)
- [uint8\\_t blue](#)

### 21.4.1 Field Documentation

#### blue

```
uint8_t blue  
Blue color intensity 0-0x0F
```

#### green

```
uint8_t green  
Green color intensity 0-0x0F
```

#### red

```
uint8_t red  
Red color intensity 0-0x0F  
The documentation for this struct was generated from the following file:
```

- [IncludeFiles/CCAuxTypes.h](#)

## 21.5 LedTimingType Struct Reference

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

### Data Fields

- [uint8\\_t onTime](#)
- [uint8\\_t offTime](#)
- [uint8\\_t idleTime](#)
- [uint8\\_t nrOfPulses](#)

### 21.5.1 Field Documentation

#### idleTime

```
uint8_t idleTime  
LED idle time in 100ms
```

**nrOfPulses**

`uint8_t` nrOfPulses  
Pulses per sequences

**offTime**

`uint8_t` offTime  
LED off time in 10ms

**onTime**

`uint8_t` onTime  
LED on time in 10ms

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

- [IncludeFiles/CCAuxTypes.h](#)

## 21.6 received\_video Struct Reference

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

### Data Fields

- [uint16\\_t](#) received\_width
- [uint16\\_t](#) received\_height
- [uint8\\_t](#) received\_framerate

### 21.6.1 Field Documentation

**received\_framerate**

`uint8_t` received\_framerate

**received\_height**

`uint16_t` received\_height

**received\_width**

`uint16_t` received\_width

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

- [IncludeFiles/CCAuxTypes.h](#)

## 21.7 TimerType Struct Reference

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

### Data Fields

- [uint32\\_t TotRunTime](#)
- [uint32\\_t TotSuspTime](#)
- [uint32\\_t TotHeatTime](#)
- [uint32\\_t RunTime40\\_60](#)
- [uint32\\_t RunTime60\\_70](#)
- [uint32\\_t RunTime70\\_80](#)
- [uint32\\_t Above80RunTime](#)

### 21.7.1 Detailed Description

Diagnostic timer data

### 21.7.2 Field Documentation

#### Above80RunTime

[uint32\\_t](#) Above80RunTime  
Total runtime in 70-80deg (minutes)

#### RunTime40\_60

[uint32\\_t](#) RunTime40\_60  
Total heating time (minutes)

#### RunTime60\_70

[uint32\\_t](#) RunTime60\_70  
Total runtime in 40-60deg (minutes)

#### RunTime70\_80

[uint32\\_t](#) RunTime70\_80  
Total runtime in 60-70deg (minutes)

#### TotHeatTime

[uint32\\_t](#) TotHeatTime  
Total suspend time (minutes)

#### TotRunTime

[uint32\\_t](#) TotRunTime



### TotSuspTime

`uint32_t` TotSuspTime

Total running time (minutes)

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

- [IncludeFiles/CCAuxTypes.h](#)

## 21.8 UpgradeStatus Struct Reference

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

### Data Fields

- enum [UpgradeAction](#) currentAction
- [uint8\\_t](#) percent
- [eErr](#) errorCode

### 21.8.1 Detailed Description

Upgrade Status

### 21.8.2 Field Documentation

#### currentAction

enum [UpgradeAction](#) currentAction

#### errorCode

[eErr](#) errorCode

Represents the percentage of completion of the current action

#### percent

`uint8_t` percent

The current action.

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

- [IncludeFiles/CCAuxTypes.h](#)

## 21.9 version\_info Struct Reference

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

## Data Fields

- [uint8\\_t major](#)
- [uint8\\_t minor](#)
- [uint8\\_t release](#)
- [uint8\\_t build](#)

### 21.9.1 Field Documentation

#### build

`uint8_t build`  
version build number

#### major

`uint8_t major`  
version major number

#### minor

`uint8_t minor`  
version minor number

#### release

`uint8_t release`  
version release number  
The documentation for this struct was generated from the following file:

- [IncludeFiles/CCAuxTypes.h](#)

## 21.10 video\_dec\_command Struct Reference

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

### Data Fields

- [uint8\\_t decoder\\_register](#)
- [uint8\\_t register\\_value](#)

#### 21.10.1 Field Documentation

##### decoder\_register

`uint8_t decoder_register`

**register\_value**

`uint8_t` register\_value

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

- [IncludeFiles/CCAuxTypes.h](#)

# Chapter 22

## File Documentation

### 22.1 IncludeFiles/About.h File Reference

#### Namespaces

- [CrossControl](#)

#### Typedefs

- typedef void \* [ABOUTHANDLE](#)

#### Functions

- EXTERN\_C CCAUXDLL\_API ABOUTHANDLE CCAUXDLL\_CALLING\_CONV [GetAbout](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [About\\_release](#) (ABOUTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainPCBSerial](#) (ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getUnitSerial](#) (ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainPCBArt](#) (ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) length)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainManufacturingDate](#) (ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainHWversion](#) (ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainProdRev](#) (ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getMainProdArtNr](#) (ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfETHConnections](#) (ABOUTHANDLE, [uint8\\_t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfCANConnections](#) (ABOUTHANDLE, [uint8\\_t](#) \*NrOfConnections)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfVideoConnections](#)  
(ABOUTHANDLE, [uint8\\_t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfUSBConnections](#)  
(ABOUTHANDLE, [uint8\\_t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfSerialConnections](#)  
(ABOUTHANDLE, [uint8\\_t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfDigIOConnections](#)  
(ABOUTHANDLE, [uint8\\_t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsDisplayAvailable](#)  
(ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsTouchScreenAvailable](#)  
(ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsPCBBluetoothAvailable](#)  
(ABOUTHANDLE, bool \*available)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getDisplayResolution](#)  
(ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnPCBSerial](#)  
(ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnPCBArt](#)  
(ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) length)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnManufacturingDate](#)  
(ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getAddOnHWversion](#)  
(ABOUTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsWLANMounted](#)  
(ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsGPSSMounted](#)  
(ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsGPRSMounted](#)  
(ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsBTMounted](#)  
(ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getFrontPcbRev](#)  
(ABOUTHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsIOExpanderMounted](#)  
(ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIOExpanderValue](#)  
(ABOUTHANDLE, [uint16\\_t](#) \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_hasOsBooted](#)  
(ABOUTHANDLE, bool \*bootComplete)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getIsAnybusMounted](#)  
(ABOUTHANDLE, bool \*mounted)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfCfgInConnections](#)  
(ABOUTHANDLE, [uint8\\_t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfPWMOutConnections](#)  
(ABOUTHANDLE, [uint8\\_t](#) \*NrOfConnections)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfButtons](#)  
(ABOUTHANDLE, [int32\\_t](#) \*numbuttons)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getNrOfAnalogInputs](#) (ABOUTHANDLE, [int32\\_t](#) \*numanalogins)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_getUserEepromData](#) (ABOUTHANDLE, [char\\_t](#) \*buff, [uint16\\_t](#) length)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [About\\_setUserEepromData](#) (ABOUTHANDLE, [uint16\\_t](#) startpos, const [char\\_t](#) \*buff, [uint16\\_t](#) length)

## 22.2 IncludeFiles/Adc.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [ADCHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API ADCHANDLE CCAUXDLL\_CALLING\_CONV [GetAdc](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Adc\\_release](#) (ADCHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Adc\\_getVoltage](#) (ADCHANDLE, VoltageEnum selection, [float64\\_t](#) \*value)

## 22.3 IncludeFiles/AuxVersion.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [AUXVERSIONHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API AUXVERSIONHANDLE CCAUXDLL\_CALLING\_CONV [GetAuxVersion](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [AuxVersion\\_release](#) (AUXVERSIONHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getFPGAVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getSSVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getFrontVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getCCAuxVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getOSVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getCCAuxDrvVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getCanFdMcuVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getCanFdApiVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getFullIOSVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build, [char\\_t](#) \*tag, [uint32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [AuxVersion\\_getFullCCAuxVersion](#) (AUXVERSIONHANDLE, [uint8\\_t](#) \*major, [uint8\\_t](#) \*minor, [uint8\\_t](#) \*release, [uint8\\_t](#) \*build, [char\\_t](#) \*tag, [uint32\\_t](#) len)

## 22.4 IncludeFiles/Backlight.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [BACKLIGHTHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API BACKLIGHTHANDLE CCAUXDLL\_CALLING\_CONV [GetBacklight](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Backlight\\_release](#) (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_getIntensity](#) (BACKLIGHTHANDLE, [uint8\\_t](#) \*intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_setIntensity](#) (BACKLIGHTHANDLE, [uint8\\_t](#) intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_getStatus](#) (BACKLIGHTHANDLE, [uint8\\_t](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_getHWStatus](#) (BACKLIGHTHANDLE, bool \*status)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_startAutomaticBL](#) (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_stopAutomaticBL](#) (BACKLIGHTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_getAutomaticBLStatus](#) (BACKLIGHTHANDLE, [uint8\\_t](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_setAutomaticBLParams](#) (BACKLIGHTHANDLE, bool bSoftTransitions)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_getAutomaticBLParams](#) (BACKLIGHTHANDLE, bool \*bSoftTransitions, [float64\\_t](#) \*k)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_setAutomaticBLFilter](#) (BACKLIGHTHANDLE, [uint32\\_t](#) averageWndSize, [uint32\\_t](#) rejectWndSize, [uint32\\_t](#) rejectDeltaInLux, LightSensorSamplingMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_getAutomaticBLFilter](#) (BACKLIGHTHANDLE, [uint32\\_t](#) \*averageWndSize, [uint32\\_t](#) \*rejectWndSize, [uint32\\_t](#) \*rejectDeltaInLux, LightSensorSamplingMode \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_getLedDimming](#) (BACKLIGHTHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Backlight\\_setLedDimming](#) (BACKLIGHTHANDLE, CCStatus status)

## 22.5 IncludeFiles/Battery.h File Reference

### Data Structures

- struct [BatteryTimerType](#)

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [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)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Battery\\_release](#) (BATTERYHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_isBatteryPresent](#) (BATTERYHANDLE, bool \*batteryIsPresent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatteryVoltageStatus](#) (BATTERYHANDLE, [uint8\\_t](#) \*batteryVoltagePercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatteryChargingStatus](#) (BATTERYHANDLE, ChargingStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getPowerSource](#) (BATTERYHANDLE, PowerSource \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatteryTemp](#) (BATTERYHANDLE, [int16\\_t](#) \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getHwErrorStatus](#) (BATTERYHANDLE, ErrorStatus \*errorCode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getTimer](#) (BATTERYHANDLE, BatteryTimerType \*times)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getMinMaxTemp](#) (BATTERYHANDLE, [int16\\_t](#) \*minTemp, [int16\\_t](#) \*maxTemp)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatteryHWversion](#) (BATTERYHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatterySwVersion](#) (BATTERYHANDLE, [uint16\\_t](#) \*major, [uint16\\_t](#) \*minor, [uint16\\_t](#) \*release, [uint16\\_t](#) \*build)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Battery\\_getBatterySerial](#) (BATTERYHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)

## 22.6 IncludeFiles/Buzzer.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [BUZZERHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API BUZZERHANDLE CCAUXDLL\_CALLING↔\_CONV [GetBuzzer](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Buzzer\\_release](#) (BUZZERHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_getFrequency](#) (BUZZERHANDLE, [uint16\\_t](#) \*frequency)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_getVolume](#) (BUZZERHANDLE, [uint16\\_t](#) \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_getScaledVolume](#) (BUZZERHANDLE, [uint8\\_t](#) \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_getTrigger](#) (BUZZERHANDLE, bool \*trigger)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_setFrequency](#) (BUZZERHANDLE, [uint16\\_t](#) frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_setVolume](#) (BUZZERHANDLE, [uint16\\_t](#) volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_setScaledVolume](#) (BUZZERHANDLE, [uint8\\_t](#) volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_setTrigger](#) (BUZZERHANDLE, bool trigger)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Buzzer\\_buzze](#) (BUZZERHANDLE, [int32\\_t](#) time, bool blocking)

## 22.7 IncludeFiles/CanSetting.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [CANSETTINGHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API CANSETTINGHANDLE CCAUXDLL\_CALLING↔\_CONV [GetCanSetting](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [CanSetting\\_release](#) (CANSETTINGHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CanSetting\\_getBaudrate](#) (CANSETTINGHANDLE, [uint8\\_t](#) net, [uint16\\_t](#) \*baudrate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CanSetting\\_getFrameType](#) (CANSETTINGHANDLE, [uint8\\_t](#) net, CanFrameType \*frameType)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CanSetting\\_setBaudrate](#) (CANSETTINGHANDLE, [uint8\\_t](#) net, [uint16\\_t](#) baudrate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CanSetting\\_setFrameType](#) (CANSETTINGHANDLE, [uint8\\_t](#) net, CanFrameType frameType)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CanSetting\\_getFlexibleDataBaudrates](#) (CANSETTINGHANDLE, [uint8\\_t](#) net, [uint16\\_t](#) \*baudrateNominal, [uint16\\_t](#) \*baudrate↔Data)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CanSetting\\_setFlexibleDataBaudrates](#) (CANSETTINGHANDLE, [uint8\\_t](#) net, [uint16\\_t](#) baudrateNominal, [uint16\\_t](#) baudrate↔Data)

## 22.8 IncludeFiles/CCAuxErrors.h File Reference

### Namespaces

- [CrossControl](#)

### Functions

- EXTERN\_C CCAUXDLL\_API [char\\_t](#) const \*CCAUXDLL\_CALLING\_CONV [GetErrorStringA](#) (eErr errCode)
- EXTERN\_C CCAUXDLL\_API [wchar\\_t](#) const \*CCAUXDLL\_CALLING\_CONV [GetErrorStringW](#) (eErr errCode)

## 22.9 IncludeFiles/CCAuxTypes.h File Reference

### Data Structures

- struct [received\\_video](#)
- struct [video\\_dec\\_command](#)
- struct [version\\_info](#)
- struct [BuzzerSetup](#)
- struct [LedTimingType](#)
- struct [FpgaLedTimingType](#)
- struct [LedColorMixType](#)
- struct [TimerType](#)
- struct [UpgradeStatus](#)

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef float [float32\\_t](#)
- typedef double [float64\\_t](#)
- typedef char [char\\_t](#)
- typedef signed char [int8\\_t](#)
- typedef signed short [int16\\_t](#)
- typedef signed int [int32\\_t](#)
- typedef unsigned char [uint8\\_t](#)
- typedef unsigned short [uint16\\_t](#)
- typedef unsigned int [uint32\\_t](#)
- typedef signed \_\_int64 [int64\\_t](#)
- typedef unsigned \_\_int64 [uint64\\_t](#)
- typedef struct version\_info [VersionType](#)

## Enumerations

- enum `VoltageEnum` {  
`VOLTAGE_24VIN = 0`, `VOLTAGE_24V`, `VOLTAGE_12V`, `VOLTAGE_12VID`  
, `VOLTAGE_5V`, `VOLTAGE_3V3`, `VOLTAGE_VTFT`, `VOLTAGE_5VSTB`,  
`VOLTAGE_1V9`, `VOLTAGE_1V8`, `VOLTAGE_1V5`, `VOLTAGE_1V2`,  
`VOLTAGE_1V05`, `VOLTAGE_1V0`, `VOLTAGE_0V9`, `VOLTAGE_VREF_INT`,  
`VOLTAGE_24V_BACKUP`, `VOLTAGE_2V5`, `VOLTAGE_1V1`, `VOLTAGE_1V3_PER`  
, `VOLTAGE_1V3_VDDA`, `VOLTAGE_3V3STBY`, `VOLTAGE_VPMIC`, `VOLTAGE_VMAIN`  
, `VOLTAGE_UB`, `VOLTAGE_AI_1`, `VOLTAGE_AI_2`, `VOLTAGE_AI_3`,  
`VOLTAGE_5V_MP`, `VOLTAGE_12V_BL`, `VOLTAGE_VHEAT`, `VOLTAGE_VIN_FILTER`  
, `VOLTAGE_VIN_PROT`, `VOLTAGE_BL_ANODE`, `VOLTAGE_IGN_STATE`,  
`VOLTAGE_IGN_VALUE`,  
`VOLTAGE_PRE_IGN_STATE`, `VOLTAGE_PRE_IGN_VALUE`, `VOLTAGE_VBACKUP`  
, `VOLTAGE_END` }
- enum `LightSensorOperationRange` { `RangeStandard = 0`, `RangeExtended = 1` }
- enum `LightSensorSamplingMode` { `SamplingModeStandard = 0`, `SamplingModeExtended`  
, `SamplingModeAuto` }
- enum `CCStatus` { `Disabled = 0`, `Enabled = 1` }
- enum `eErr` {  
`ERR_SUCCESS = 0`, `ERR_OPEN_FAILED = 1`, `ERR_NOT_SUPPORTED = 2`  
, `ERR_UNKNOWN_FEATURE = 3`,  
`ERR_DATATYPE_MISMATCH = 4`, `ERR_CODE_NOT_EXIST = 5`, `ERR_BUFFER_SIZE`  
= `6`, `ERR_IOCTL_FAILED = 7`,  
`ERR_INVALID_DATA = 8`, `ERR_INVALID_PARAMETER = 9`, `ERR_CREATE_THREAD`  
= `10`, `ERR_IN_PROGRESS = 11`,  
`ERR_CHECKSUM = 12`, `ERR_INIT_FAILED = 13`, `ERR_VERIFY_FAILED =`  
`14`, `ERR_DEVICE_READ_DATA_FAILED = 15`,  
`ERR_DEVICE_WRITE_DATA_FAILED = 16`, `ERR_COMMAND_FAILED =`  
`17`, `ERR_EEPROM = 18`, `ERR_JIDA_TEMP = 19`,  
`ERR_AVERAGE_CALC_STARTED = 20`, `ERR_NOT_RUNNING = 21`, `ERR_I2C_EXPANDER_READ_FAILED`  
= `22`, `ERR_I2C_EXPANDER_WRITE_FAILED = 23`,  
`ERR_I2C_EXPANDER_INIT_FAILED = 24`, `ERR_NEWER_SS_VERSION_REQUIRED`  
= `25`, `ERR_NEWER_FPGA_VERSION_REQUIRED = 26`, `ERR_NEWER_FRONT_VERSION_REQUIRED`  
= `27`,  
`ERR_TELEMATICS_GPRS_NOT_AVAILABLE = 28`, `ERR_TELEMATICS_WLAN_NOT_AVAILABLE`  
= `29`, `ERR_TELEMATICS_BT_NOT_AVAILABLE = 30`, `ERR_TELEMATICS_GPS_NOT_AVAILABLE`  
= `31`,  
`ERR_MEM_ALLOC_FAIL = 32`, `ERR_JOIN_THREAD = 33`, `ERR_INVALID_STARTUP_TRIGGER`  
= `34`, `ERR_END` }
- 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`,  
`Analog_channel_END` }

- enum `videoStandard` {  
`STD_MJ_NTSC = 0`, `STD_B_D_G_H_I_N_PAL = 1`, `STD_M_PAL = 2`, `STD_PAL = 3`,  
`STD_NTSC = 4`, `STD_SECAM = 5` }
- enum `VideoRotation` { `RotNone = 0`, `Rot90`, `Rot180`, `Rot270` }
- enum `CanFrameType` { `FrameStandard`, `FrameExtended`, `FrameStandardExtended` }
- enum `CanMode` { `CanModeClassic`, `CanModeFlexibleData` }
- enum `TriggerConf` {  
`Front_Button_Enabled = 1`, `OnOff_Signal_Enabled = 2`, `Both_Button_And_Signal_Enabled = 3`,  
`CAN_Button_Activity = 5`, `CAN_OnOff_Activity = 6`, `CAN_Button_OnOff_Activity = 7`, `CI_Button_Activity = 9`,  
`CI_OnOff_Activity = 10`, `CI_Button_OnOff_Activity = 11`, `CI_CAN_Button_Activity = 13`, `CI_CAN_OnOff_Activity = 14`,  
`All_Events = 15`, `Last_trigger_conf` }
- 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`, `BPTS_END` }
- enum `OCDStatus` { `OCD_OK = 0`, `OCD_OC = 1`, `OCD_POWER_OFF = 2` }
- 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 `RS4XXPort` { `RS4XXPort1 = 1`, `RS4XXPort2`, `RS4XXPort3`, `RS4XXPort4` }
- enum `CfgInModeEnum` {  
`CFGIN_NOT_IN_USE = 0`, `CFGIN_HI_SWITCH`, `CFGIN_LOW_SWITCH`,  
`CFGIN_VOLTAGE_2V5`,  
`CFGIN_VOLTAGE_5V`, `CFGIN_RESISTANCE`, `CFGIN_FREQ_FLOATING`,  
`CFGIN_FREQ_PULLUP`,  
`CFGIN_FREQ_PULLDOWN`, `CFGIN_RESISTANCE_500`, `CFGIN_CURRENT_4_20`,  
`CFGIN_VOLTAGE_10V`,  
`CFGIN_VOLTAGE_32V`, `CFGIN_DIGITAL_PD_5V`, `CFGIN_DIGITAL_PD_10V`,  
`CFGIN_DIGITAL_PD_32V`,  
`CFGIN_DIGITAL_F_5V`, `CFGIN_DIGITAL_F_10V`, `CFGIN_DIGITAL_F_32V`,  
`CFGIN_DIGITAL_PU_5V`,

```

    CFGIN_DIGITAL_PU_10V , CFGIN_DIGITAL_PU_32V , CFGIN_FREQ_PD_5V
    , CFGIN_FREQ_PD_10V ,
    CFGIN_FREQ_PD_32V , CFGIN_FREQ_F_5V , CFGIN_FREQ_F_10V , CFGIN_FREQ_F_32V
    ,
    CFGIN_FREQ_PU_5V , CFGIN_FREQ_PU_10V , CFGIN_FREQ_PU_32V , CFGIN_VS_FreqInMode1
    ,
    CFGIN_VS_FreqInMode1PU , CFGIN_VS_FreqInMode2 , CFGIN_VS_FreqInMode2PU
    , CFGIN_MAX }
• enum ButtonConfigEnum {
    BUTTON_ONLY_MP_ACTION = 0x00 , BUTTON_AS_STARTUP_TRIG = 0x02
    , BUTTON_AS_ACTION_TRIG = 0x04 , BUTTON_AS_ACTION_STARTUP_TRIG
    = 0x06 ,
    BUTTON_AS_BACKLIGHT_DECREASE = 0x08 , BUTTON_AS_BACKLIGHT_DECR_STARTUP_TRIG
    = 0x0A , BUTTON_AS_BACKLIGHT_INCREASE = 0x0C , BUTTON_AS_BACKLIGHT_INCR_STARTUP_TRIG
    = 0x0E }
• enum BootModeEnum {
    BOOTMODE_EMMC = 0 , BOOTMODE_SD , BOOTMODE_SERIAL , BOOTMODE_RESCUE
    ,
    BOOTMODE_RESCUE_SPECIAL }
• enum ConfigOnOffTriggerMode { CONFIG_ONOFF_EDGE_TRIGGER = 0 ,
    CONFIG_ONOFF_LEVEL_TRIGGER }
• enum PowerOutput {
    PowerOutput1 = 0 , PowerOutput2 , PowerOutput3 , PowerOutput4 ,
    PowerOutput5 , PowerOutput6 , PowerOutputMax }
• enum SystemMode {
    SYSTEMMODE_Startup = 0 , SYSTEMMODE_StartupRescue = 1 , SYSTEMMODE_StartupRescueFactoryRes
    = 2 , SYSTEMMODE_NormalRunning = 3 ,
    SYSTEMMODE_RescueRunning = 4 , SYSTEMMODE_RescueRunningFactoryReset
    = 5 , SYSTEMMODE_Unknown = 6 }

```

### 22.9.1 Typedef Documentation

#### **char\_t**

```
typedef char char_t
```

#### **float32\_t**

```
typedef float float32_t
```

#### **float64\_t**

```
typedef double float64_t
```

**int16\_t**

```
typedef signed short int16\_t
```

**int32\_t**

```
typedef signed int int32\_t
```

**int64\_t**

```
typedef long int int64\_t
```

**int8\_t**

```
typedef signed char int8\_t
```

**uint16\_t**

```
typedef unsigned short uint16\_t
```

**uint32\_t**

```
typedef unsigned int uint32\_t
```

**uint64\_t**

```
typedef unsigned long int uint64\_t
```

**uint8\_t**

```
typedef unsigned char uint8\_t
```

## 22.10 IncludeFiles/CCPlatform.h File Reference

## 22.11 IncludeFiles/CfgIn.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [CFGINHANDLE](#)

## Functions

- EXTERN\_C CCAUXDLL\_API CFGINHANDLE CCAUXDLL\_CALLING\_↔  
CONV [GetCfgIn](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [CfgIn\\_release](#)  
(CFGINHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CfgIn\\_setCfgInMode](#)  
(CFGINHANDLE, [uint8\\_t](#) channel, CfgInModeEnum set\_mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CfgIn\\_getCfgInMode](#)  
(CFGINHANDLE, [uint8\\_t](#) channel, CfgInModeEnum \*get\_mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CfgIn\\_getValue](#)  
(CFGINHANDLE, [uint8\\_t](#) channel, [uint16\\_t](#) \*sample\_value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CfgIn\\_getPwmValue](#)  
(CFGINHANDLE, [uint8\\_t](#) channel, [float32\\_t](#) \*frequency, [uint8\\_t](#) \*duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CfgIn\\_getFrequencyValue](#)  
(CFGINHANDLE, [uint8\\_t](#) channel, [float32\\_t](#) \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CfgIn\\_getMinFrequencyThreshold](#)  
(CFGINHANDLE, [uint8\\_t](#) channel, [float32\\_t](#) \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CfgIn\\_setMinFrequencyThreshold](#)  
(CFGINHANDLE, [uint8\\_t](#) channel, [float32\\_t](#) frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [CfgIn\\_setFrequencyFilterLevel](#)  
(CFGINHANDLE, [uint8\\_t](#) level)

## 22.12 IncludeFiles/Config.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [CONFIGHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API CONFIGHANDLE CCAUXDLL\_CALLING\_↔  
\_CONV [GetConfig](#) ()
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Config\\_release](#)  
(CONFIGHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getStartupTriggerConfig](#)  
(CONFIGHANDLE, TriggerConf \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getShortButtonPressAction](#)  
(CONFIGHANDLE, PowerAction \*action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getLongButtonPressAction](#)  
(CONFIGHANDLE, PowerAction \*action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getOnOffSigAction](#)  
(CONFIGHANDLE, PowerAction \*action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getFrontBtnTrigTime](#)  
(CONFIGHANDLE, [uint16\\_t](#) \*triggertime)



- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getExtOnOffSigTrigTime](#) (CONFIGHANDLE, [uint32\\_t](#) \*triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getButtonFunction](#) (CONFIGHANDLE, [uint8\\_t](#) button\_number, ButtonConfigEnum \*button\_config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getSuspendMaxTime](#) (CONFIGHANDLE, [uint16\\_t](#) \*maxTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getCanStartupPowerConfig](#) (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getVideoStartupPowerConfig](#) (CONFIGHANDLE, [uint8\\_t](#) \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getExtFanStartupPowerConfig](#) (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getStartupVoltageConfig](#) (CONFIGHANDLE, [float64\\_t](#) \*voltage)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getHeatingTempLimit](#) (CONFIGHANDLE, [int16\\_t](#) \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getPowerOnStartup](#) (CONFIGHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setStartupTriggerConfig](#) (CONFIGHANDLE, TriggerConf conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setShortButtonPressAction](#) (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setLongButtonPressAction](#) (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setOnOffSigAction](#) (CONFIGHANDLE, PowerAction action)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setFrontBtnTrigTime](#) (CONFIGHANDLE, [uint16\\_t](#) triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setExtOnOffSigTrigTime](#) (CONFIGHANDLE, [uint32\\_t](#) triggertime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setButtonFunction](#) (CONFIGHANDLE, [uint8\\_t](#) button\_number, ButtonConfigEnum button\_config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setSuspendMaxTime](#) (CONFIGHANDLE, [uint16\\_t](#) maxTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setCanStartupPowerConfig](#) (CONFIGHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setVideoStartupPowerConfig](#) (CONFIGHANDLE, [uint8\\_t](#) config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setExtFanStartupPowerConfig](#) (CONFIGHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setStartupVoltageConfig](#) (CONFIGHANDLE, [float64\\_t](#) voltage)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setHeatingTempLimit](#) (CONFIGHANDLE, [int16\\_t](#) temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setPowerOnStartup](#) (CONFIGHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setRS485Enabled](#) (CONFIGHANDLE, RS4XXPort port, bool enabled)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getRS485Enabled](#) (CONFIGHANDLE, RS4XXPort port, bool \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setOnOffTriggerMode](#) (CONFIGHANDLE, ConfigOnOffTriggerMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getOnOffTriggerMode](#) (CONFIGHANDLE, ConfigOnOffTriggerMode \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setNextBootMode](#) (CONFIGHANDLE, BootModeEnum mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getNextBootMode](#) (CONFIGHANDLE, BootModeEnum \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setOSAliveMonitoring](#) (CONFIGHANDLE, CCStatus enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getOSAliveMonitoring](#) (CONFIGHANDLE, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getOnOffSignalState](#) (CONFIGHANDLE, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getDigPowerOutputStartupConfig](#) (CONFIGHANDLE, PowerOutput output, CCStatus \*enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setDigPowerOutputStartupConfig](#) (CONFIGHANDLE, PowerOutput output, CCStatus enabled)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getShortBeepSettings](#) (CONFIGHANDLE, uint16\_t \*duration, uint16\_t \*frequency, uint16\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setShortBeepSettings](#) (CONFIGHANDLE, uint16\_t duration, uint16\_t frequency, uint16\_t volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_getLongBeepSettings](#) (CONFIGHANDLE, uint16\_t \*duration, uint16\_t \*frequency, uint16\_t \*volume)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Config\\_setLongBeepSettings](#) (CONFIGHANDLE, uint16\_t duration, uint16\_t frequency, uint16\_t volume)

## Variables

- const [uint8\\_t Video1Conf](#) = (1 << 0)
- const [uint8\\_t Video2Conf](#) = (1 << 1)
- const [uint8\\_t Video3Conf](#) = (1 << 2)
- const [uint8\\_t Video4Conf](#) = (1 << 3)

## 22.13 IncludeFiles/Diagnostic.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [DIAGNOSTICHANDLE](#)

## Functions

- EXTERN\_C CCAUXDLL\_API DIAGNOSTICHANDLE CCAUXDLL\_CALLING\_CONV GetDiagnostic (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Diagnostic\_release (DIAGNOSTICHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getSSTemp (DIAGNOSTICHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPCBTemp (DIAGNOSTICHANDLE, int16\_t \*temperature)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPMTemp (DIAGNOSTICHANDLE, uint8\_t index, int16\_t \*temperature, JidaSensorType \*jst)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getStartupReason (DIAGNOSTICHANDLE, uint16\_t \*reason)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getShutDownReason (DIAGNOSTICHANDLE, uint16\_t \*reason)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getHwErrorStatus (DIAGNOSTICHANDLE, uint16\_t \*errorCode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getTimer (DIAGNOSTICHANDLE, TimerType \*times)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getMinMaxTemp (DIAGNOSTICHANDLE, int16\_t \*minTemp, int16\_t \*maxTemp)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_getPowerCycles (DIAGNOSTICHANDLE, uint16\_t \*powerCycles)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Diagnostic\_clearHwErrorStatus (DIAGNOSTICHANDLE)

## 22.14 IncludeFiles/DiagnosticCodes.h File Reference

### Namespaces

- [CrossControl](#)

### Enumerations

- enum [startupReasonCodes](#) {  
[startupReasonCodeUndefined](#) = 0x0000 , [startupReasonCodeButtonPress](#) = 0x0055  
, [startupReasonCodeExtCtrl](#) = 0x00AA , [startupReasonCodeMPRestart](#) = 0x00F0 ,  
[startupReasonCodePowerOnStartup](#) = 0x000F , [startupReasonCodeCanActivity](#)  
= 0x003c , [startupReasonCodeCIActivity](#) = 0x00c3 , [startupReasonAlwaysStart](#)  
= 0x00e1 ,  
[startupReasonUnknownTrigger](#) = 0x001e }
- enum [shutDownReasonCodes](#) { [shutdownReasonCodeNoError](#) = 0x001F }
- enum [hwErrorStatusCodes](#) { [errCodeNoErr](#) = 0 }

## Functions

- EXTERN\_C CCAUXDLL\_API [char\\_t](#) const \*CCAUXDLL\_CALLING\_CONV [GetHwErrorStatusStringA](#) ([uint16\\_t](#) errCode)
- EXTERN\_C CCAUXDLL\_API [wchar\\_t](#) const \*CCAUXDLL\_CALLING\_CONV [GetHwErrorStatusStringW](#) ([uint16\\_t](#) errCode)
- EXTERN\_C CCAUXDLL\_API [char\\_t](#) const \*CCAUXDLL\_CALLING\_CONV [GetStartupReasonStringA](#) ([uint16\\_t](#) code)
- EXTERN\_C CCAUXDLL\_API [wchar\\_t](#) const \*CCAUXDLL\_CALLING\_CONV [GetStartupReasonStringW](#) ([uint16\\_t](#) code)

## 22.15 IncludeFiles/DigIO.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [DIGIOHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API [DIGIOHANDLE](#) CCAUXDLL\_CALLING\_CONV [GetDigIO](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [DigIO\\_release](#) ([DIGIOHANDLE](#))
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [DigIO\\_getDigIO](#) ([DIGIOHANDLE](#), [uint8\\_t](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [DigIO\\_setDigIO](#) ([DIGIOHANDLE](#), [uint8\\_t](#) state)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [DigIO\\_getDigPowerOutput](#) ([DIGIOHANDLE](#), [PowerOutput](#) output, [CCStatus](#) \*enabled, [uint8\\_t](#) \*status)
- EXTERN\_C CCAUXDLL\_API [eErr](#) CCAUXDLL\_CALLING\_CONV [DigIO\\_setDigPowerOutput](#) ([DIGIOHANDLE](#), [PowerOutput](#) output, [CCStatus](#) enabled)

### Variables

- const [uint8\\_t](#) [DigitalIn\\_1](#) = (1 << 0)
- const [uint8\\_t](#) [DigitalIn\\_2](#) = (1 << 1)
- const [uint8\\_t](#) [DigitalIn\\_3](#) = (1 << 2)
- const [uint8\\_t](#) [DigitalIn\\_4](#) = (1 << 3)

## 22.16 IncludeFiles/FirmwareUpgrade.h File Reference

### Namespaces

- [CrossControl](#)

**Typedefs**

- typedef void \* [FIRMWAREUPGHANDLE](#)

**Functions**

- EXTERN\_C CCAUXDLL\_API FIRMWAREUPGHANDLE CCAUXDLL\_CALLING↔\_CONV [GetFirmwareUpgrade](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_release](#) (FIRMWAREUPGHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startFpgaUpgrade](#) (FIRMWAREUPGHANDLE, const [char\\_t](#) \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startFpgaVerification](#) (FIRMWAREUPGHANDLE, const [char\\_t](#) \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startSSUpgrade](#) (FIRMWAREUPGHANDLE, const [char\\_t](#) \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startSSVerification](#) (FIRMWAREUPGHANDLE, const [char\\_t](#) \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startFrontUpgrade](#) (FIRMWAREUPGHANDLE, const [char\\_t](#) \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startFrontVerification](#) (FIRMWAREUPGHANDLE, const [char\\_t](#) \*filename, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startDMCUUpgrade](#) (FIRMWAREUPGHANDLE, const [char\\_t](#) \*filename, bool blocking, [uint8\\_t](#) index)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_startDMCUVerification](#) (FIRMWAREUPGHANDLE, const [char\\_t](#) \*filename, bool blocking, [uint8\\_t](#) index)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_getUpgradeStatus](#) (FIRMWAREUPGHANDLE, UpgradeStatus \*status, bool blocking)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FirmwareUpgrade\\_shutDown](#) (FIRMWAREUPGHANDLE)

**22.17 IncludeFiles/FrontLED.h File Reference****Namespaces**

- [CrossControl](#)

**Typedefs**

- typedef void \* [FRONTLEDHANDLE](#)

**Functions**

- EXTERN\_C CCAUXDLL\_API FRONTLEDHANDLE CCAUXDLL\_CALLING↔\_CONV [GetFrontLED](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [FrontLED\\_release](#) (FRONTLEDHANDLE)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getSignal](#) (FRONTLEDHANDLE, [float64\\_t](#) \*frequency, [uint8\\_t](#) \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getOnTime](#) (FRONTLEDHANDLE, [uint8\\_t](#) \*onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getOffTime](#) (FRONTLEDHANDLE, [uint8\\_t](#) \*offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getIdleTime](#) (FRONTLEDHANDLE, [uint8\\_t](#) \*idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getNrOfPulses](#) (FRONTLEDHANDLE, [uint8\\_t](#) \*nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getColor](#) (FRONTLEDHANDLE, [uint8\\_t](#) \*red, [uint8\\_t](#) \*green, [uint8\\_t](#) \*blue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getStandardColor](#) (FRONTLEDHANDLE, [CCAuxColor](#) \*color)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getEnabledDuringStartup](#) (FRONTLEDHANDLE, [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setSignal](#) (FRONTLEDHANDLE, [float64\\_t](#) frequency, [uint8\\_t](#) dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setOnTime](#) (FRONTLEDHANDLE, [uint8\\_t](#) onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setOffTime](#) (FRONTLEDHANDLE, [uint8\\_t](#) offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setIdleTime](#) (FRONTLEDHANDLE, [uint8\\_t](#) idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setNrOfPulses](#) (FRONTLEDHANDLE, [uint8\\_t](#) nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setColor](#) (FRONTLEDHANDLE, [uint8\\_t](#) red, [uint8\\_t](#) green, [uint8\\_t](#) blue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setStandardColor](#) (FRONTLEDHANDLE, [CCAuxColor](#) color)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setOff](#) (FRONTLEDHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setEnabledDuringStartup](#) (FRONTLEDHANDLE, [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getBootLEDConfig](#) (FRONTLEDHANDLE, [uint8\\_t](#) \*red, [uint8\\_t](#) \*green, [uint8\\_t](#) \*blue, [float32\\_t](#) \*frequency, [uint8\\_t](#) \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setBootLEDConfig](#) (FRONTLEDHANDLE, [uint8\\_t](#) red, [uint8\\_t](#) green, [uint8\\_t](#) blue, [float32\\_t](#) frequency, [uint8\\_t](#) dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_getPostBootLEDConfig](#) (FRONTLEDHANDLE, [uint8\\_t](#) \*red, [uint8\\_t](#) \*green, [uint8\\_t](#) \*blue, [float32\\_t](#) \*frequency, [uint8\\_t](#) \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [FrontLED\\_setPostBootLEDConfig](#) (FRONTLEDHANDLE, [uint8\\_t](#) red, [uint8\\_t](#) green, [uint8\\_t](#) blue, [float32\\_t](#) frequency, [uint8\\_t](#) dutyCycle)

## 22.18 IncludeFiles/Lightsensor.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [LIGHTSENSORHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API LIGHTSENSORHANDLE CCAUXDLL\_CALLING\_CONV [GetLightsensor](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Lightsensor\\_release](#) (LIGHTSENSORHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_getIlluminance](#) (LIGHTSENSORHANDLE, [uint16\\_t](#) \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_getIlluminance2](#) (LIGHTSENSORHANDLE, [uint16\\_t](#) \*value, [uint8\\_t](#) \*ch0, [uint8\\_t](#) \*ch1)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_getAverageIlluminance](#) (LIGHTSENSORHANDLE, [uint16\\_t](#) \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_startAverageCalc](#) (LIGHTSENSORHANDLE, [uint32\\_t](#) averageWndSize, [uint32\\_t](#) rejectWndSize, [uint32\\_t](#) rejectDeltaInLux, LightSensorSamplingMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_stopAverageCalc](#) (LIGHTSENSORHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_getOperatingRange](#) (LIGHTSENSORHANDLE, LightSensorOperationRange \*range)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Lightsensor\\_setOperatingRange](#) (LIGHTSENSORHANDLE, LightSensorOperationRange range)

## 22.19 IncludeFiles/Power.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [POWERHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API POWERHANDLE CCAUXDLL\_CALLING\_CONV [GetPower](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Power\\_release](#) (POWERHANDLE)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getBLPowerStatus](#) (POWERHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getCanPowerStatus](#) (POWERHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getVideoPowerStatus](#) (POWERHANDLE, uint8\_t \*videoStatus)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getExtFanPowerStatus](#) (POWERHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getButtonPowerTransitionStatus](#) (POWERHANDLE, ButtonPowerTransitionStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getVideoOCDStatus](#) (POWERHANDLE, OCDStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_getCanOCDStatus](#) (POWERHANDLE, OCDStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_setBLPowerStatus](#) (POWERHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_setCanPowerStatus](#) (POWERHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_setVideoPowerStatus](#) (POWERHANDLE, uint8\_t status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_setExtFanPowerStatus](#) (POWERHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Power\\_ackPowerRequest](#) (POWERHANDLE)

## 22.20 IncludeFiles/PowerMgr.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [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)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [PowerMgr\\_release](#) (POWERMGRHANDLE)



- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PowerMgr\\_registerControlledSuspendOrS](#)  
(POWERMGRHANDLE, PowerMgrConf conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PowerMgr\\_getConfiguration](#)  
(POWERMGRHANDLE, PowerMgrConf \*conf)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PowerMgr\\_getPowerMgrStatus](#)  
(POWERMGRHANDLE, PowerMgrStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PowerMgr\\_setAppReadyForSuspendOrS](#)  
(POWERMGRHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PowerMgr\\_hasResumed](#)  
(POWERMGRHANDLE, bool \*resumed)

## 22.21 IncludeFiles/PWMOut.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [PWMOUTHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API PWMOUTHANDLE CCAUXDLL\_CALLING\_CONV [GetPWMOut](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [PWMOut\\_release](#)  
(PWMOUTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PWMOut\\_setPWMOutputChannelDutyC](#)  
(PWMOUTHANDLE, uint8\_t channel, uint8\_t duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PWMOut\\_setPWMOutputChannelFreque](#)  
(PWMOUTHANDLE, uint8\_t channel, float32\_t frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PWMOut\\_getPWMOutputChannelDutyC](#)  
(PWMOUTHANDLE, uint8\_t channel, uint8\_t \*duty\_cycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PWMOut\\_getPWMOutputChannelFreque](#)  
(PWMOUTHANDLE, uint8\_t channel, float32\_t \*frequency)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PWMOut\\_getPWMOutputStatus](#)  
(PWMOUTHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [PWMOut\\_setPWMOutOff](#)  
(PWMOUTHANDLE, uint8\_t channel)

## 22.22 IncludeFiles/Releasenotes.dox File Reference

## 22.23 IncludeFiles/Smart.h File Reference

### Namespaces

- [CrossControl](#)

## Typedefs

- typedef void \* [SMARTHANDLE](#)

## Functions

- EXTERN\_C CCAUXDLL\_API SMARTHANDLE CCAUXDLL\_CALLING\_↔  
\_CONV [GetSmart](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [Smart\\_release](#)  
(SMARTHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Smart\\_getRemainingLifeTime](#)  
(SMARTHANDLE, [uint8\\_t](#) \*lifetimepercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Smart\\_getRemainingLifeTime2](#)  
(SMARTHANDLE, [uint8\\_t](#) \*lifetimepercent)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Smart\\_getDeviceSerial](#)  
(SMARTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Smart\\_getDeviceSerial2](#)  
(SMARTHANDLE, [char\\_t](#) \*buff, [int32\\_t](#) len)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Smart\\_getInitialTime](#)  
(SMARTHANDLE, [time\\_t](#) \*time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Smart\\_getInitialTime2](#)  
(SMARTHANDLE, [time\\_t](#) \*time)

## 22.24 IncludeFiles/SoftKey.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [SOFTKEYHANDLE](#)

### Functions

- EXTERN\_C CCAUXDLL\_API SOFTKEYHANDLE CCAUXDLL\_CALLING\_↔  
\_CONV [GetSoftKey](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [SoftKey\\_release](#)  
(SOFTKEYHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getStatus](#)  
(SOFTKEYHANDLE, [uint16\\_t](#) \*value)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightIntensity](#)  
(SOFTKEYHANDLE, [uint8\\_t](#) key, [uint8\\_t](#) intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightIntensity](#)  
(SOFTKEYHANDLE, [uint8\\_t](#) key, [uint8\\_t](#) \*intensity)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setMultipleBacklightIntensities](#)  
(SOFTKEYHANDLE, [uint8\\_t](#) \*keys, [uint8\\_t](#) \*intensities, [uint8\\_t](#) array\_size)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getMultipleBacklightIntensities](#)  
(SOFTKEYHANDLE, [uint8\\_t](#) \*keys, [uint8\\_t](#) \*intensities, [uint8\\_t](#) array\_size)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightSignal](#) (SOFTKEYHANDLE, [float64\\_t](#) \*frequency, [uint8\\_t](#) \*dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightOnTime](#) (SOFTKEYHANDLE, [uint8\\_t](#) \*onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightOffTime](#) (SOFTKEYHANDLE, [uint8\\_t](#) \*offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightIdleTime](#) (SOFTKEYHANDLE, [uint8\\_t](#) \*idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightNrOfPulses](#) (SOFTKEYHANDLE, [uint8\\_t](#) \*nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightSignal](#) (SOFTKEYHANDLE, [float64\\_t](#) frequency, [uint8\\_t](#) dutyCycle)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightOnTime](#) (SOFTKEYHANDLE, [uint8\\_t](#) onTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightOffTime](#) (SOFTKEYHANDLE, [uint8\\_t](#) offTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightIdleTime](#) (SOFTKEYHANDLE, [uint8\\_t](#) idleTime)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightNrOfPulses](#) (SOFTKEYHANDLE, [uint8\\_t](#) nrOfPulses)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightOff](#) (SOFTKEYHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBacklightEnabledDuringStart](#) (SOFTKEYHANDLE, [CCStatus](#) \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBacklightEnabledDuringStart](#) (SOFTKEYHANDLE, [CCStatus](#) status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_getBootBacklightConfig](#) (SOFTKEYHANDLE, [uint8\\_t](#) \*bootIntensity, [float32\\_t](#) \*bootFrequency, [uint8\\_t](#) \*bootDutyCycle, [uint8\\_t](#) \*postBootIntensity, [float32\\_t](#) \*postBootFrequency, [uint8\\_t](#) \*postBootDutyCycle, [CCStatus](#) \*postBootConfig)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [SoftKey\\_setBootBacklightConfig](#) (SOFTKEYHANDLE, [uint8\\_t](#) bootIntensity, [float32\\_t](#) bootFrequency, [uint8\\_t](#) boot↔DutyCycle, [uint8\\_t](#) postBootIntensity, [float32\\_t](#) postBootFrequency, [uint8\\_t](#) post↔BootDutyCycle, [CCStatus](#) postBootConfig)

## 22.25 IncludeFiles/Telematics.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [TELEMATICSHANDLE](#)

## Functions

- EXTERN\_C CCAUXDLL\_API TELEMATICSHANDLE CCAUXDLL\_CALLING\_CONV GetTelematics (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Telematics\_release (TELEMATICSHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getTelematicsAvailable (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPRSPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPRSStartupPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getWLANPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getWLANStartupPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getBTPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getBTStartupPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPSPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPSStartupPowerStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_getGPSAntennaStatus (TELEMATICSHANDLE, CCStatus \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setGPRSPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setGPRSStartupPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setWLANPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setWLANStartupPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setBTPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setBTStartupPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setGPSPowerStatus (TELEMATICSHANDLE, CCStatus status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Telematics\_setGPSStartupPowerStatus (TELEMATICSHANDLE, CCStatus status)

## 22.26 IncludeFiles/TouchScreen.h File Reference

### Namespaces

- [CrossControl](#)

## Typedefs

- typedef void \* [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 , [TS\\_TCHAUTOCAL](#) = 12 }

## Functions

- EXTERN\_C CCAUXDLL\_API TOUCHSCREENHANDLE CCAUXDLL\_CALLING\_CONV [GetTouchScreen](#) (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV [TouchScreen\\_release](#) (TOUCHSCREENHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getMode](#) (TOUCHSCREENHANDLE, TouchScreenModeSettings \*config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getMouseRightClickTime](#) (TOUCHSCREENHANDLE, [uint16\\_t](#) \*time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setMode](#) (TOUCHSCREENHANDLE, TouchScreenModeSettings config)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setMouseRightClickTime](#) (TOUCHSCREENHANDLE, [uint16\\_t](#) time)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_setAdvancedSetting](#) (TOUCHSCREENHANDLE, TSAdvancedSettingsParameter param, [uint16\\_t](#) data)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [TouchScreen\\_getAdvancedSetting](#) (TOUCHSCREENHANDLE, TSAdvancedSettingsParameter param, [uint16\\_t](#) \*data)

## 22.27 IncludeFiles/TouchScreenCalib.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [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)`
- `EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV TouchScreenCalib\_release`  
`(TOUCHSCREENCALIBHANDLE)`
- `EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV TouchScreenCalib\_setMode`  
`(TOUCHSCREENCALIBHANDLE, CalibrationModeSettings mode)`
- `EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV TouchScreenCalib\_getMode`  
`(TOUCHSCREENCALIBHANDLE, CalibrationModeSettings *mode)`
- `EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV TouchScreenCalib\_setCalibrationPoint`  
`(TOUCHSCREENCALIBHANDLE, uint8\_t pointNr)`
- `EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV TouchScreenCalib\_checkCalibrationPoint`  
`(TOUCHSCREENCALIBHANDLE, bool *finished, uint8\_t pointNr)`
- `EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV TouchScreenCalib\_getConfigParam`  
`(TOUCHSCREENCALIBHANDLE, CalibrationConfigParam param, uint16\_t`  
`*value)`
- `EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV TouchScreenCalib\_setConfigParam`  
`(TOUCHSCREENCALIBHANDLE, CalibrationConfigParam param, uint16\_t`  
`value)`
- `EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV TouchScreenCalib\_autoSensorCalib`  
`(TOUCHSCREENCALIBHANDLE)`

## 22.28 IncludeFiles/Video.h File Reference

### Namespaces

- [CrossControl](#)

### Typedefs

- typedef void \* [VIDEOHANDLE](#)

## Functions

- EXTERN\_C CCAUXDLL\_API VIDEOHANDLE CCAUXDLL\_CALLING\_CONV GetVideo (void)
- EXTERN\_C CCAUXDLL\_API void CCAUXDLL\_CALLING\_CONV Video\_release (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_init (VIDEOHANDLE, uint8\_t deviceNr)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_showVideo (VIDEOHANDLE, bool show)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setDeInterlaceMode (VIDEOHANDLE, DeInterlaceMode mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getDeInterlaceMode (VIDEOHANDLE, DeInterlaceMode \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setMirroring (VIDEOHANDLE, CCStatus mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getMirroring (VIDEOHANDLE, CCStatus \*mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setRotation (VIDEOHANDLE, VideoRotation rotation)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getRotation (VIDEOHANDLE, VideoRotation \*rotation)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setActiveChannel (VIDEOHANDLE, VideoChannel channel)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getActiveChannel (VIDEOHANDLE, VideoChannel \*channel)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setColorKeys (VIDEOHANDLE, uint8\_t rKey, uint8\_t gKey, uint8\_t bKey)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getColorKeys (VIDEOHANDLE, uint8\_t \*rKey, uint8\_t \*gKey, uint8\_t \*bKey)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setVideoArea (VIDEOHANDLE, uint16\_t topLeftX, uint16\_t topLeftY, uint16\_t bottomRightX, uint16\_t bottomRightY)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getRawImage (VIDEOHANDLE, uint16\_t \*width, uint16\_t \*height, float32\_t \*frameRate)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getVideoArea (VIDEOHANDLE, uint16\_t \*topLeftX, uint16\_t \*topLeftY, uint16\_t \*bottomRigthX, uint16\_t \*bottomRigthY)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getVideoStandard (VIDEOHANDLE, videoStandard \*standard)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getStatus (VIDEOHANDLE, uint8\_t \*status)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_setScaling (VIDEOHANDLE, float32\_t x, float32\_t y)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_getScaling (VIDEOHANDLE, float32\_t \*x, float32\_t \*y)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV Video\_activateSnapshot (VIDEOHANDLE, bool activate)

- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_takeSnapshot](#) (VIDEOHANDLE, const [char\\_t](#) \*path, bool bInterlaced)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_takeSnapshotRaw](#) (VIDEOHANDLE, [char\\_t](#) \*rawImgBuffer, [uint32\\_t](#) rawImgBuffSize, bool bInterlaced)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_takeSnapshotBmp](#) (VIDEOHANDLE, [char\\_t](#) \*\*bmpBuffer, [uint32\\_t](#) \*bmpBufSize, bool bInterlaced, bool bNTSCFormat)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_createBitmap](#) (VIDEOHANDLE, [char\\_t](#) \*\*bmpBuffer, [uint32\\_t](#) \*bmpBufSize, const [char\\_t](#) \*rawImgBuffer, [uint32\\_t](#) rawImgBuffSize, bool bInterlaced, bool bNTSCFormat)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_freeBmpBuffer](#) (VIDEOHANDLE, [char\\_t](#) \*bmpBuffer)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_minimize](#) (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_restore](#) (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_setDecoderReg](#) (VIDEOHANDLE, [uint8\\_t](#) decoderRegister, [uint8\\_t](#) registerValue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_getDecoderReg](#) (VIDEOHANDLE, [uint8\\_t](#) decoderRegister, [uint8\\_t](#) \*registerValue)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_setCropping](#) (VIDEOHANDLE, [uint8\\_t](#) top, [uint8\\_t](#) left, [uint8\\_t](#) bottom, [uint8\\_t](#) right)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_getCropping](#) (VIDEOHANDLE, [uint8\\_t](#) \*top, [uint8\\_t](#) \*left, [uint8\\_t](#) \*bottom, [uint8\\_t](#) \*right)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_showFrame](#) (VIDEOHANDLE)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_setGraphicsOverlay](#) (VIDEOHANDLE, CCStatus mode)
- EXTERN\_C CCAUXDLL\_API eErr CCAUXDLL\_CALLING\_CONV [Video\\_getGraphicsOverlay](#) (VIDEOHANDLE, CCStatus \*mode)



# Index

- About functions, 31
  - About\_getAddOnHWversion, 33
  - About\_getAddOnManufacturingDate, 33
  - About\_getAddOnPCBArt, 34
  - About\_getAddOnPCBSerial, 34
  - About\_getDisplayResolution, 35
  - About\_getFrontPcbRev, 35
  - About\_getIOExpanderValue, 36
  - About\_getIsAnybusMounted, 36
  - About\_getIsBTMounted, 36
  - About\_getIsDisplayAvailable, 37
  - About\_getIsGPRSMounted, 37
  - About\_getIsGPSMounted, 38
  - About\_getIsIOExpanderMounted, 38
  - About\_getIsPCBBluetoothAvailable, 40
  - About\_getIsTouchScreenAvailable, 40
  - About\_getIsWLANMounted, 41
  - About\_getMainHWversion, 41
  - About\_getMainManufacturingDate, 42
  - About\_getMainPCBArt, 42
  - About\_getMainPCBSerial, 43
  - About\_getMainProdArtNr, 43
  - About\_getMainProdRev, 44
  - About\_getNrOfAnalogInputs, 44
  - About\_getNrOfButtons, 45
  - About\_getNrOfCANConnections, 45
  - About\_getNrOfCfgInConnections, 46
  - About\_getNrOfDigIOConnections, 46
  - About\_getNrOfETHConnections, 47
  - About\_getNrOfPWMOutConnections, 47
  - About\_getNrOfSerialConnections, 48
  - About\_getNrOfUSBConnections, 48
  - About\_getNrOfVideoConnections, 49
  - About\_getUnitSerial, 49
  - About\_getUserEepromData, 50
  - About\_hasOsBooted, 50
  - About\_release, 51
  - About\_setUserEepromData, 51
  - GetAbout, 52
  - About\_getAddOnHWversion
    - About functions, 33
  - About\_getAddOnManufacturingDate
    - About functions, 33
  - About\_getAddOnPCBArt
    - About functions, 34
  - About\_getAddOnPCBSerial
    - About functions, 34
  - About\_getDisplayResolution
    - About functions, 35
  - About\_getFrontPcbRev
    - About functions, 35
  - About\_getIOExpanderValue
    - About functions, 36
  - About\_getIsAnybusMounted
    - About functions, 36
  - About\_getIsBTMounted
    - About functions, 36
  - About\_getIsDisplayAvailable
    - About functions, 37
  - About\_getIsGPRSMounted
    - About functions, 37
  - About\_getIsGPSMounted
    - About functions, 38
  - About\_getIsIOExpanderMounted
    - About functions, 38
  - About\_getIsPCBBluetoothAvailable
    - About functions, 40
  - About\_getIsTouchScreenAvailable
    - About functions, 40
  - About\_getIsWLANMounted
    - About functions, 41
  - About\_getMainHWversion
    - About functions, 41
  - About\_getMainManufacturingDate
    - About functions, 42
  - About\_getMainPCBArt
    - About functions, 42
  - About\_getMainPCBSerial
    - About functions, 43
  - About\_getMainProdArtNr
    - About functions, 43

- About functions, 43
- About\_getMainProdRev
  - About functions, 44
- About\_getNrOfAnalogInputs
  - About functions, 44
- About\_getNrOfButtons
  - About functions, 45
- About\_getNrOfCANConnections
  - About functions, 45
- About\_getNrOfCfgInConnections
  - About functions, 46
- About\_getNrOfDigIOConnections
  - About functions, 46
- About\_getNrOfETHConnections
  - About functions, 47
- About\_getNrOfPWMOutConnections
  - About functions, 47
- About\_getNrOfSerialConnections
  - About functions, 48
- About\_getNrOfUSBConnections
  - About functions, 48
- About\_getNrOfVideoConnections
  - About functions, 49
- About\_getUnitSerial
  - About functions, 49
- About\_getUserEepromData
  - About functions, 50
- About\_hasOsBooted
  - About functions, 50
- About\_release
  - About functions, 51
- About\_setUserEepromData
  - About functions, 51
- ABOUTHANDLE
  - CrossControl, 265
- Above80RunTime
  - TimerType, 296
- ActionShutDown
  - CrossControl, 278
- ActionSuspend
  - CrossControl, 278
- Adc functions, 52
  - Adc\_getVoltage, 52
  - Adc\_release, 54
  - GetAdc, 55
- Adc\_getVoltage
  - Adc functions, 52
- Adc\_release
  - Adc functions, 54
- ADCHANDLE
  - CrossControl, 265
- All\_Events
  - CrossControl, 282
- Analog\_Channel\_1
  - CrossControl, 284
- Analog\_Channel\_2
  - CrossControl, 284
- Analog\_Channel\_3
  - CrossControl, 284
- Analog\_Channel\_4
  - CrossControl, 284
- Analog\_channel\_END
  - CrossControl, 284
- ApplicationControlled
  - CrossControl, 278
- AuxVersion functions, 56
  - AuxVersion\_getCanFdApiVersion, 56
  - AuxVersion\_getCanFdMcuVersion, 57
  - AuxVersion\_getCCAuxDrvVersion, 58
  - AuxVersion\_getCCAuxVersion, 59
  - AuxVersion\_getFPGAVersion, 59
  - AuxVersion\_getFrontVersion, 60
  - AuxVersion\_getFullCCAuxVersion, 61
  - AuxVersion\_getFullOSVersion, 62
  - AuxVersion\_getOSVersion, 62
  - AuxVersion\_getSSVersion, 63
  - AuxVersion\_release, 64
  - GetAuxVersion, 64
- AuxVersion\_getCanFdApiVersion
  - AuxVersion functions, 56
- AuxVersion\_getCanFdMcuVersion
  - AuxVersion functions, 57
- AuxVersion\_getCCAuxDrvVersion
  - AuxVersion functions, 58
- AuxVersion\_getCCAuxVersion
  - AuxVersion functions, 59
- AuxVersion\_getFPGAVersion
  - AuxVersion functions, 59
- AuxVersion\_getFrontVersion
  - AuxVersion functions, 60
- AuxVersion\_getFullCCAuxVersion
  - AuxVersion functions, 61
- AuxVersion\_getFullOSVersion
  - AuxVersion functions, 62
- AuxVersion\_getOSVersion
  - AuxVersion functions, 62
- AuxVersion\_getSSVersion
  - AuxVersion functions, 63
- AuxVersion\_release
  - AuxVersion functions, 64

- AUXVERSIONHANDLE
  - CrossControl, 265
- Backlight functions, 65
  - Backlight\_getAutomaticBLFilter, 65
  - Backlight\_getAutomaticBLParams, 66
  - Backlight\_getAutomaticBLStatus, 67
  - Backlight\_getHWStatus, 67
  - Backlight\_getIntensity, 68
  - Backlight\_getLedDimming, 68
  - Backlight\_getStatus, 69
  - Backlight\_release, 69
  - Backlight\_setAutomaticBLFilter, 70
  - Backlight\_setAutomaticBLParams, 70
  - Backlight\_setIntensity, 71
  - Backlight\_setLedDimming, 71
  - Backlight\_startAutomaticBL, 72
  - Backlight\_stopAutomaticBL, 72
  - GetBacklight, 72
- Backlight\_getAutomaticBLFilter
  - Backlight functions, 65
- Backlight\_getAutomaticBLParams
  - Backlight functions, 66
- Backlight\_getAutomaticBLStatus
  - Backlight functions, 67
- Backlight\_getHWStatus
  - Backlight functions, 67
- Backlight\_getIntensity
  - Backlight functions, 68
- Backlight\_getLedDimming
  - Backlight functions, 68
- Backlight\_getStatus
  - Backlight functions, 69
- Backlight\_release
  - Backlight functions, 69
- Backlight\_setAutomaticBLFilter
  - Backlight functions, 70
- Backlight\_setAutomaticBLParams
  - Backlight functions, 70
- Backlight\_setIntensity
  - Backlight functions, 71
- Backlight\_setLedDimming
  - Backlight functions, 71
- Backlight\_startAutomaticBL
  - Backlight functions, 72
- Backlight\_stopAutomaticBL
  - Backlight functions, 72
- BACKLIGHTHANDLE
  - CrossControl, 265
- Battery functions, 73
  - Battery\_getBatteryChargingStatus, 74
  - Battery\_getBatteryHWversion, 75
  - Battery\_getBatterySerial, 75
  - Battery\_getBatterySwVersion, 76
  - Battery\_getBatteryTemp, 77
  - Battery\_getBatteryVoltageStatus, 77
  - Battery\_getHwErrorStatus, 78
  - Battery\_getMinMaxTemp, 79
  - Battery\_getPowerSource, 80
  - Battery\_getTimer, 80
  - Battery\_isBatteryPresent, 81
  - Battery\_release, 82
  - GetBattery, 82
  - Battery\_getBatteryChargingStatus
    - Battery functions, 74
  - Battery\_getBatteryHWversion
    - Battery functions, 75
  - Battery\_getBatterySerial
    - Battery functions, 75
  - Battery\_getBatterySwVersion
    - Battery functions, 76
  - Battery\_getBatteryTemp
    - Battery functions, 77
  - Battery\_getBatteryVoltageStatus
    - Battery functions, 77
  - Battery\_getHwErrorStatus
    - Battery functions, 78
  - Battery\_getMinMaxTemp
    - Battery functions, 79
  - Battery\_getPowerSource
    - Battery functions, 80
  - Battery\_getTimer
    - Battery functions, 80
  - Battery\_isBatteryPresent
    - Battery functions, 81
  - Battery\_release
    - Battery functions, 82
- BATTERYHANDLE
  - CrossControl, 266
- BatterySuspend
  - CrossControl, 278
- BatteryTimerType, 291
  - RunTime\_0\_40, 291
  - RunTime\_40\_60, 291
  - RunTime\_60\_70, 291
  - RunTime\_70\_80, 291
  - RunTime\_Above80, 292
  - RunTime\_m20, 292
  - RunTime\_m20\_0, 292
  - TotRunTimeBattery, 292

- TotRunTimeMain, 292
- BLUE
  - CrossControl, 271
- blue
  - LedColorMixType, 294
- BOOTMODE\_EMMC
  - CrossControl, 268
- BOOTMODE\_RESCUE
  - CrossControl, 268
- BOOTMODE\_RESCUE\_SPECIAL
  - CrossControl, 268
- BOOTMODE\_SD
  - CrossControl, 268
- BOOTMODE\_SERIAL
  - CrossControl, 268
- BootModeEnum
  - CrossControl, 268
- Both\_Button\_And\_Signal\_Enabled
  - CrossControl, 282
- BPTS\_BtnPressed
  - CrossControl, 269
- BPTS\_BtnPressedLong
  - CrossControl, 269
- BPTS\_END
  - CrossControl, 269
- BPTS\_No\_Change
  - CrossControl, 269
- BPTS\_Restart
  - CrossControl, 269
- BPTS\_ShutDown
  - CrossControl, 269
- BPTS\_SignalOff
  - CrossControl, 269
- BPTS\_Suspend
  - CrossControl, 269
- build
  - version\_info, 298
- BUTTON\_AS\_ACTION\_STARTUP\_TRIG
  - CrossControl, 268
- BUTTON\_AS\_ACTION\_TRIG
  - CrossControl, 268
- BUTTON\_AS\_BACKLIGHT\_DECR\_STARTUP\_TRIG
  - CrossControl, 268
- BUTTON\_AS\_BACKLIGHT\_DECREASE
  - CrossControl, 268
- BUTTON\_AS\_BACKLIGHT\_INCR\_STARTUP\_TRIG
  - CrossControl, 268
- BUTTON\_AS\_BACKLIGHT\_INCREASE
  - CrossControl, 268
- BUTTON\_AS\_STARTUP\_TRIG
  - CrossControl, 268
- CAN\_Button\_Activity
  - CrossControl, 268
- BUZZERHANDLE
  - CrossControl, 266
- Buzzer
  - frequency, 292
  - volume, 292
- Buzzer\_buaze
  - Buzzer functions, 83
- Buzzer\_getFrequency
  - Buzzer functions, 84
- Buzzer\_getScaledVolume
  - Buzzer functions, 85
- Buzzer\_getTrigger
  - Buzzer functions, 85
- Buzzer\_getVolume
  - Buzzer functions, 85
- Buzzer\_release
  - Buzzer functions, 86
- Buzzer\_setFrequency
  - Buzzer functions, 86
- Buzzer\_setScaledVolume
  - Buzzer functions, 87
- Buzzer\_setTrigger
  - Buzzer functions, 87
- Buzzer\_setVolume
  - Buzzer functions, 88
- Calibration
  - CalibrationConfigParam
    - CrossControl, 269
  - CalibrationModeSettings
    - CrossControl, 270

- CrossControl, 282
- CAN\_Button\_OnOff\_Activity
  - CrossControl, 282
- CAN\_OnOff\_Activity
  - CrossControl, 282
- CanFrameType
  - CrossControl, 270
- CanMode
  - CrossControl, 270
- CanModeClassic
  - CrossControl, 270
- CanModeFlexibleData
  - CrossControl, 270
- CanSetting functions, 89
  - CanSetting\_getBaudrate, 89
  - CanSetting\_getFlexibleDataBaudrates, 90
  - CanSetting\_getFrameType, 90
  - CanSetting\_release, 91
  - CanSetting\_setBaudrate, 91
  - CanSetting\_setFlexibleDataBaudrates, 92
  - CanSetting\_setFrameType, 92
  - GetCanSetting, 93
- CanSetting\_getBaudrate
  - CanSetting functions, 89
- CanSetting\_getFlexibleDataBaudrates
  - CanSetting functions, 90
- CanSetting\_getFrameType
  - CanSetting functions, 90
- CanSetting\_release
  - CanSetting functions, 91
- CanSetting\_setBaudrate
  - CanSetting functions, 91
- CanSetting\_setFlexibleDataBaudrates
  - CanSetting functions, 92
- CanSetting\_setFrameType
  - CanSetting functions, 92
- CANSETTINGHANDLE
  - CrossControl, 266
- CCAuxColor
  - CrossControl, 271
- CCAuxTypes.h
  - char\_t, 310
  - float32\_t, 310
  - float64\_t, 310
  - int16\_t, 310
  - int32\_t, 311
  - int64\_t, 311
  - int8\_t, 311
  - uint16\_t, 311
  - uint32\_t, 311
  - uint64\_t, 311
  - uint8\_t, 311
- CCStatus
  - CrossControl, 271
- CfgIn functions, 93
  - CfgIn\_getCfgInMode, 94
  - CfgIn\_getFrequencyValue, 95
  - CfgIn\_getMinFrequencyThreshold, 95
  - CfgIn\_getPwmValue, 96
  - CfgIn\_getValue, 97
  - CfgIn\_release, 98
  - CfgIn\_setCfgInMode, 99
  - CfgIn\_setFrequencyFilterLevel, 99
  - CfgIn\_setMinFrequencyThreshold, 100
  - GetCfgIn, 101
- CFGIN\_CURRENT\_4\_20
  - CrossControl, 272
- CFGIN\_DIGITAL\_F\_10V
  - CrossControl, 272
- CFGIN\_DIGITAL\_F\_32V
  - CrossControl, 272
- CFGIN\_DIGITAL\_F\_5V
  - CrossControl, 272
- CFGIN\_DIGITAL\_PD\_10V
  - CrossControl, 272
- CFGIN\_DIGITAL\_PD\_32V
  - CrossControl, 272
- CFGIN\_DIGITAL\_PD\_5V
  - CrossControl, 272
- CFGIN\_DIGITAL\_PU\_10V
  - CrossControl, 272
- CFGIN\_DIGITAL\_PU\_32V
  - CrossControl, 272
- CFGIN\_DIGITAL\_PU\_5V
  - CrossControl, 272
- CFGIN\_FREQ\_F\_10V
  - CrossControl, 273
- CFGIN\_FREQ\_F\_32V
  - CrossControl, 273
- CFGIN\_FREQ\_F\_5V
  - CrossControl, 272
- CFGIN\_FREQ\_FLOATING
  - CrossControl, 272
- CFGIN\_FREQ\_PD\_10V
  - CrossControl, 272
- CFGIN\_FREQ\_PD\_32V
  - CrossControl, 272
- CFGIN\_FREQ\_PD\_5V

- CrossControl, [272](#)
- CFGIN\_FREQ\_PU\_10V
  - CrossControl, [273](#)
- CFGIN\_FREQ\_PU\_32V
  - CrossControl, [273](#)
- CFGIN\_FREQ\_PU\_5V
  - CrossControl, [273](#)
- CFGIN\_FREQ\_PULLDOWN
  - CrossControl, [272](#)
- CFGIN\_FREQ\_PULLUP
  - CrossControl, [272](#)
- CfgIn.getCfgInMode
  - CfgIn functions, [94](#)
- CfgIn.getFrequencyValue
  - CfgIn functions, [95](#)
- CfgIn.getMinFrequencyThreshold
  - CfgIn functions, [95](#)
- CfgIn.getPwmValue
  - CfgIn functions, [96](#)
- CfgIn.getValue
  - CfgIn functions, [97](#)
- CFGIN\_HI\_SWITCH
  - CrossControl, [271](#)
- CFGIN\_LOW\_SWITCH
  - CrossControl, [271](#)
- CFGIN\_MAX
  - CrossControl, [273](#)
- CFGIN\_NOT\_IN\_USE
  - CrossControl, [271](#)
- CfgIn.release
  - CfgIn functions, [98](#)
- CFGIN\_RESISTANCE
  - CrossControl, [271](#)
- CFGIN\_RESISTANCE\_500
  - CrossControl, [272](#)
- CfgIn.setCfgInMode
  - CfgIn functions, [99](#)
- CfgIn.setFrequencyFilterLevel
  - CfgIn functions, [99](#)
- CfgIn.setMinFrequencyThreshold
  - CfgIn functions, [100](#)
- CFGIN\_VOLTAGE\_10V
  - CrossControl, [272](#)
- CFGIN\_VOLTAGE\_2V5
  - CrossControl, [271](#)
- CFGIN\_VOLTAGE\_32V
  - CrossControl, [272](#)
- CFGIN\_VOLTAGE\_5V
  - CrossControl, [271](#)
- CFGIN\_VS\_FreqInMode1
  - CrossControl, [273](#)
- CFGIN\_VS\_FreqInMode1PU
  - CrossControl, [273](#)
- CFGIN\_VS\_FreqInMode2
  - CrossControl, [273](#)
- CFGIN\_VS\_FreqInMode2PU
  - CrossControl, [273](#)
- CFGINHANDLE
  - CrossControl, [266](#)
- CfgInModeEnum
  - CrossControl, [271](#)
- char\_t
  - CCAuxTypes.h, [310](#)
- ChargingStatus
  - CrossControl, [273](#)
- ChargingStatus.Charging
  - CrossControl, [273](#)
- ChargingStatus.FullyCharged
  - CrossControl, [273](#)
- ChargingStatus.NoCharge
  - CrossControl, [273](#)
- ChargingStatus.TempHigh
  - CrossControl, [273](#)
- ChargingStatus.TempLow
  - CrossControl, [273](#)
- ChargingStatus.Unknown
  - CrossControl, [274](#)
- CI\_Button\_Activity
  - CrossControl, [282](#)
- CI\_Button\_OnOff\_Activity
  - CrossControl, [282](#)
- CI\_CAN\_Button\_Activity
  - CrossControl, [282](#)
- CI\_CAN\_OnOff\_Activity
  - CrossControl, [282](#)
- CI\_OnOff\_Activity
  - CrossControl, [282](#)
- Config functions, [101](#)
  - Config.getButtonFunction, [103](#)
  - Config.getCanStartupPowerConfig, [104](#)
  - Config.getDigPowerOutputStartupConfig, [105](#)
  - Config.getExtFanStartupPowerConfig, [105](#)
  - Config.getExtOnOffSigTrigTime, [105](#)
  - Config.getFrontBtnTrigTime, [106](#)
  - Config.getHeatingTempLimit, [106](#)
  - Config.getLongBeepSettings, [107](#)
  - Config.getLongButtonPressAction, [107](#)
  - Config.getNextBootMode, [108](#)

- Config\_getOnOffSigAction, 108
- Config\_getOnOffSignalState, 109
- Config\_getOnOffTriggerMode, 109
- Config\_getOSAliveMonitoring, 109
- Config\_getPowerOnStartup, 110
- Config\_getRS485Enabled, 110
- Config\_getShortBeepSettings, 111
- Config\_getShortButtonPressAction, 111
- Config\_getStartupTriggerConfig, 112
- Config\_getStartupVoltageConfig, 113
- Config\_getSuspendMaxTime, 113
- Config\_getVideoStartupPowerConfig, 114
- Config\_release, 114
- Config\_setButtonFunction, 114
- Config\_setCanStartupPowerConfig, 116
- Config\_setDigPowerOutputStartupConfig, 116
- Config\_setExtFanStartupPowerConfig, 117
- Config\_setExtOnOffSigTrigTime, 117
- Config\_setFrontBtnTrigTime, 117
- Config\_setHeatingTempLimit, 118
- Config\_setLongBeepSettings, 118
- Config\_setLongButtonPressAction, 119
- Config\_setNextBootMode, 119
- Config\_setOnOffSigAction, 120
- Config\_setOnOffTriggerMode, 120
- Config\_setOSAliveMonitoring, 121
- Config\_setPowerOnStartup, 121
- Config\_setRS485Enabled, 121
- Config\_setShortBeepSettings, 122
- Config\_setShortButtonPressAction, 123
- Config\_setStartupTriggerConfig, 123
- Config\_setStartupVoltageConfig, 124
- Config\_setSuspendMaxTime, 124
- Config\_setVideoStartupPowerConfig, 124
- GetConfig, 125
- CONFIG\_13P\_CALIBRATION\_POINT\_BORDER
  - CrossControl, 269
- CONFIG\_13P\_CALIBRATION\_TRANSITION\_MAX
  - CrossControl, 270
- CONFIG\_13P\_CALIBRATION\_TRANSITION\_MIN
  - CrossControl, 270
- CONFIG\_5P\_CALIBRATION\_POINT\_BORDER
  - CrossControl, 269
- CONFIG\_CALIBRATION\_MEASUREMENTS
  - CrossControl, 269
- CONFIG\_CALIBRATION\_WITH
  - CrossControl, 269
- CrossControl, 269
- Config\_getButtonFunction
  - Config functions, 103
- Config\_getCanStartupPowerConfig
  - Config functions, 104
- Config\_getDigPowerOutputStartupConfig
  - Config functions, 105
- Config\_getExtFanStartupPowerConfig
  - Config functions, 105
- Config\_getExtOnOffSigTrigTime
  - Config functions, 105
- Config\_getFrontBtnTrigTime
  - Config functions, 106
- Config\_getHeatingTempLimit
  - Config functions, 106
- Config\_getLongBeepSettings
  - Config functions, 107
- Config\_getLongButtonPressAction
  - Config functions, 107
- Config\_getNextBootMode
  - Config functions, 108
- Config\_getOnOffSigAction
  - Config functions, 108
- Config\_getOnOffSignalState
  - Config functions, 109
- Config\_getOnOffTriggerMode
  - Config functions, 109
- Config\_getOSAliveMonitoring
  - Config functions, 109
- Config\_getPowerOnStartup
  - Config functions, 110
- Config\_getRS485Enabled
  - Config functions, 110
- Config\_getShortBeepSettings
  - Config functions, 111
- Config\_getShortButtonPressAction
  - Config functions, 111
- Config\_getStartupTriggerConfig
  - Config functions, 112
- Config\_getStartupVoltageConfig
  - Config functions, 113
- Config\_getSuspendMaxTime
  - Config functions, 113
- Config\_getVideoStartupPowerConfig
  - Config functions, 114
- CONFIG\_ONOFF\_EDGE\_TRIGGER
  - CrossControl, 274
- CONFIG\_ONOFF\_LEVEL\_TRIGGER
  - CrossControl, 274
- Config\_release

- Config functions, 114
- Config.setButtonFunction
  - Config functions, 114
- Config.setCanStartupPowerConfig
  - Config functions, 116
- Config.setDigPowerOutputStartupConfig
  - Config functions, 116
- Config.setExtFanStartupPowerConfig
  - Config functions, 117
- Config.setExtOnOffSigTrigTime
  - Config functions, 117
- Config.setFrontBtnTrigTime
  - Config functions, 117
- Config.setHeatingTempLimit
  - Config functions, 118
- Config.setLongBeepSettings
  - Config functions, 118
- Config.setLongButtonPressAction
  - Config functions, 119
- Config.setNextBootMode
  - Config functions, 119
- Config.setOnOffSigAction
  - Config functions, 120
- Config.setOnOffTriggerMode
  - Config functions, 120
- Config.setOSAliveMonitoring
  - Config functions, 121
- Config.setPowerOnStartup
  - Config functions, 121
- Config.setRS485Enabled
  - Config functions, 121
- Config.setShortBeepSettings
  - Config functions, 122
- Config.setShortButtonPressAction
  - Config functions, 123
- Config.setStartupTriggerConfig
  - Config functions, 123
- Config.setStartupVoltageConfig
  - Config functions, 124
- Config.setSuspendMaxTime
  - Config functions, 124
- Config.setVideoStartupPowerConfig
  - Config functions, 124
- CONFIGHANDLE
  - CrossControl, 266
- ConfigOnOffTriggerMode
  - CrossControl, 274
- CrossControl, 245
  - ABOUTHANDLE, 265
  - ActionShutDown, 278
  - ActionSuspend, 278
  - ADCHANDLE, 265
  - All\_Events, 282
  - Analog\_Channel\_1, 284
  - Analog\_Channel\_2, 284
  - Analog\_Channel\_3, 284
  - Analog\_Channel\_4, 284
  - Analog\_channel\_END, 284
  - ApplicationControlled, 278
  - AUXVERSIONHANDLE, 265
  - BACKLIGHTHANDLE, 265
  - BATTERYHANDLE, 266
  - BatterySuspend, 278
  - BLUE, 271
  - BOOTMODE\_EMMC, 268
  - BOOTMODE\_RESCUE, 268
  - BOOTMODE\_RESCUE\_SPECIAL, 268
  - BOOTMODE\_SD, 268
  - BOOTMODE\_SERIAL, 268
  - BootModeEnum, 268
  - Both\_Button\_And\_Signal\_Enabled, 282
  - BPTS\_BtnPressed, 269
  - BPTS\_BtnPressedLong, 269
  - BPTS\_END, 269
  - BPTS\_No\_Change, 269
  - BPTS\_Restart, 269
  - BPTS\_ShutDown, 269
  - BPTS\_SignalOff, 269
  - BPTS\_Suspend, 269
  - BUTTON\_AS\_ACTION\_STARTUP\_TRIG, 268
  - BUTTON\_AS\_ACTION\_TRIG, 268
  - BUTTON\_AS\_BACKLIGHT\_DECR\_STARTUP\_TRIG, 268
  - BUTTON\_AS\_BACKLIGHT\_DECREASE, 268
  - BUTTON\_AS\_BACKLIGHT\_INCR\_STARTUP\_TRIG, 268
  - BUTTON\_AS\_BACKLIGHT\_INCREASE, 268
  - BUTTON\_AS\_STARTUP\_TRIG, 268
  - BUTTON\_ONLY\_MP\_ACTION, 268
  - ButtonConfigEnum, 268
  - ButtonPowerTransitionStatus, 268
  - BUZZERHANDLE, 266
  - CalibrationConfigParam, 269
  - CalibrationModeSettings, 270
  - CAN\_Button\_Activity, 282
  - CAN\_Button\_OnOff\_Activity, 282
  - CAN\_OnOff\_Activity, 282



- CanFrameType, 270
- CanMode, 270
- CanModeClassic, 270
- CanModeFlexibleData, 270
- CANSETTINGHANDLE, 266
- CCAuxColor, 271
- CCStatus, 271
- CFGIN\_CURRENT\_4\_20, 272
- CFGIN\_DIGITAL\_F\_10V, 272
- CFGIN\_DIGITAL\_F\_32V, 272
- CFGIN\_DIGITAL\_F\_5V, 272
- CFGIN\_DIGITAL\_PD\_10V, 272
- CFGIN\_DIGITAL\_PD\_32V, 272
- CFGIN\_DIGITAL\_PD\_5V, 272
- CFGIN\_DIGITAL\_PU\_10V, 272
- CFGIN\_DIGITAL\_PU\_32V, 272
- CFGIN\_DIGITAL\_PU\_5V, 272
- CFGIN\_FREQ\_F\_10V, 273
- CFGIN\_FREQ\_F\_32V, 273
- CFGIN\_FREQ\_F\_5V, 272
- CFGIN\_FREQ\_FLOATING, 272
- CFGIN\_FREQ\_PD\_10V, 272
- CFGIN\_FREQ\_PD\_32V, 272
- CFGIN\_FREQ\_PD\_5V, 272
- CFGIN\_FREQ\_PU\_10V, 273
- CFGIN\_FREQ\_PU\_32V, 273
- CFGIN\_FREQ\_PU\_5V, 273
- CFGIN\_FREQ\_PULLDOWN, 272
- CFGIN\_FREQ\_PULLUP, 272
- CFGIN\_HI\_SWITCH, 271
- CFGIN\_LOW\_SWITCH, 271
- CFGIN\_MAX, 273
- CFGIN\_NOT\_IN\_USE, 271
- CFGIN\_RESISTANCE, 271
- CFGIN\_RESISTANCE\_500, 272
- CFGIN\_VOLTAGE\_10V, 272
- CFGIN\_VOLTAGE\_2V5, 271
- CFGIN\_VOLTAGE\_32V, 272
- CFGIN\_VOLTAGE\_5V, 271
- CFGIN\_VS\_FreqInMode1, 273
- CFGIN\_VS\_FreqInMode1PU, 273
- CFGIN\_VS\_FreqInMode2, 273
- CFGIN\_VS\_FreqInMode2PU, 273
- CFGINHANDLE, 266
- CfgInModeEnum, 271
- ChargingStatus, 273
- ChargingStatus\_Charging, 273
- ChargingStatus\_FullyCharged, 273
- ChargingStatus\_NoCharge, 273
- ChargingStatus\_TempHigh, 273
- ChargingStatus\_TempLow, 273
- ChargingStatus\_Unknown, 274
- CI\_Button\_Activity, 282
- CI\_Button\_OnOff\_Activity, 282
- CI\_CAN\_Button\_Activity, 282
- CI\_CAN\_OnOff\_Activity, 282
- CI\_OnOff\_Activity, 282
- CONFIG\_13P\_CALIBRATION\_POINT\_BORDER, 269
- CONFIG\_13P\_CALIBRATION\_TRANSITION\_MAX, 270
- CONFIG\_13P\_CALIBRATION\_TRANSITION\_MIN, 270
- CONFIG\_5P\_CALIBRATION\_POINT\_BORDER, 269
- CONFIG\_CALIBRATION\_MEASUREMENTS, 269
- CONFIG\_CALIBRATION\_WITH, 269
- CONFIG\_ONOFF\_EDGE\_TRIGGER, 274
- CONFIG\_ONOFF\_LEVEL\_TRIGGER, 274
- CONFIGHANDLE, 266
- ConfigOnOffTriggerMode, 274
- CYAN, 271
- DeInterlace\_BOB, 274
- DeInterlace\_Even, 274
- DeInterlace\_Odd, 274
- DeInterlaceMode, 274
- DIAGNOSTICHANDLE, 266
- DIGIOHANDLE, 266
- DigitalIn\_1, 289
- DigitalIn\_2, 290
- DigitalIn\_3, 290
- DigitalIn\_4, 290
- Disabled, 271
- eErr, 274
- Enabled, 271
- ERR\_AVERAGE\_CALC\_STARTED, 275
- ERR\_BUFFER\_SIZE, 275
- ERR\_CHECKSUM, 275
- ERR\_CODE\_NOT\_EXIST, 275
- ERR\_COMMAND\_FAILED, 275
- ERR\_CREATE\_THREAD, 275
- ERR\_DATATYPE\_MISMATCH, 275
- ERR\_DEVICE\_READ\_DATA\_FAILED, 275
- ERR\_DEVICE\_WRITE\_DATA\_FAILED, 275

- ERR\_EEPROM, 275
- ERR\_END, 276
- ERR\_I2C\_EXPANDER\_INIT\_FAILED, 275
- ERR\_I2C\_EXPANDER\_READ\_FAILED, 275
- ERR\_I2C\_EXPANDER\_WRITE\_FAILED, 275
- ERR\_IN\_PROGRESS, 275
- ERR\_INIT\_FAILED, 275
- ERR\_INVALID\_DATA, 275
- ERR\_INVALID\_PARAMETER, 275
- ERR\_INVALID\_STARTUP\_TRIGGER, 276
- ERR\_IOCTL\_FAILED, 275
- ERR\_JIDA\_TEMP, 275
- ERR\_JOIN\_THREAD, 276
- ERR\_MEM\_ALLOC\_FAIL, 276
- ERR\_NEWER\_FPGA\_VERSION\_REQUIRED, 275
- ERR\_NEWER\_FRONT\_VERSION\_REQUIRED, 276
- ERR\_NEWER\_SS\_VERSION\_REQUIRED, 275
- ERR\_NOT\_RUNNING, 275
- ERR\_NOT\_SUPPORTED, 274
- ERR\_OPEN\_FAILED, 274
- ERR\_SUCCESS, 274
- ERR\_TELEMATICS\_BT\_NOT\_AVAILABLE, 276
- ERR\_TELEMATICS\_GPRS\_NOT\_AVAILABLE, 276
- ERR\_TELEMATICS\_GPS\_NOT\_AVAILABLE, 276
- ERR\_TELEMATICS\_WLAN\_NOT\_AVAILABLE, 276
- ERR\_UNKNOWN\_FEATURE, 274
- ERR\_VERIFY\_FAILED, 275
- errCodeNoErr, 276
- ErrorStatus, 276
- ErrorStatus\_ChargeFail, 276
- ErrorStatus\_Init, 276
- ErrorStatus\_NoError, 276
- ErrorStatus\_Overcurrent, 276
- ErrorStatus\_SecondaryTempSensor, 276
- ErrorStatus\_ThermistorTempSensor, 276
- FIRMWAREUPGHANDLE, 266
- FrameExtended, 270
- FrameStandard, 270
- FrameStandardExtended, 270
- Front\_Button\_Enabled, 282
- FRONTLEDHANDLE, 266
- GetErrorStringA, 287
- GetErrorStringW, 288
- GetHwErrorStatusStringA, 288
- GetHwErrorStatusStringW, 288
- GetStartupReasonStringA, 289
- GetStartupReasonStringW, 289
- GREEN, 271
- hwErrorStatusCodes, 276
- JidaSensorType, 277
- Last\_trigger\_conf, 282
- LIGHTSENSORHANDLE, 266
- LightSensorOperationRange, 277
- LightSensorSamplingMode, 277
- MAGENTA, 271
- MODE\_CALIBRATION\_13P, 270
- MODE\_CALIBRATION\_5P, 270
- MODE\_CALIBRATION\_9P, 270
- MODE\_NORMAL, 270
- MODE\_UNKNOWN, 270
- MOUSE\_NEXT\_BOOT, 281
- MOUSE\_NOW, 281
- NoAction, 278
- NoRequestsPending, 279
- Normal, 278
- OCD\_OC, 278
- OCD\_OK, 278
- OCD\_POWER\_OFF, 278
- OCDStatus, 277
- Off\_Signal\_Enabled, 282
- PowerAction, 278
- POWERHANDLE, 267
- PowerMgrConf, 278
- POWERMGRHANDLE, 267
- PowerMgrStatus, 279
- PowerOutput, 279
- PowerOutput1, 279
- PowerOutput2, 279
- PowerOutput3, 279
- PowerOutput4, 279
- PowerOutput5, 279
- PowerOutput6, 279
- PowerOutputMax, 279
- PowerSource, 279
- PowerSource\_Battery, 279
- PowerSource\_ExternalPower, 279
- PWMOUTHANDLE, 267
- RangeExtended, 277
- RangeStandard, 277

- RED, 271
- Rot180, 285
- Rot270, 285
- Rot90, 285
- RotNone, 285
- RS4XXPort, 279
- RS4XXPort1, 280
- RS4XXPort2, 280
- RS4XXPort3, 280
- RS4XXPort4, 280
- SamplingModeAuto, 277
- SamplingModeExtended, 277
- SamplingModeStandard, 277
- ShutdownPending, 279
- shutdownReasonCodeNoError, 280
- shutDownReasonCodes, 280
- SMARTHANDLE, 267
- SOFTKEYHANDLE, 267
- startupReasonAlwaysStart, 280
- startupReasonCodeButtonPress, 280
- startupReasonCodeCanActivity, 280
- startupReasonCodeCIAActivity, 280
- startupReasonCodeExtCtrl, 280
- startupReasonCodeMPRestart, 280
- startupReasonCodePowerOnStartup, 280
- startupReasonCodes, 280
- startupReasonCodeUndefined, 280
- startupReasonUnknownTrigger, 280
- STD\_B\_D\_G\_H\_I\_N\_PAL, 285
- STD\_M\_J\_NTSC, 285
- STD\_M\_PAL, 285
- STD\_NTSC, 285
- STD\_PAL, 285
- STD\_SECAM, 285
- SuspendPending, 279
- SystemMode, 280
- SYSTEMMODE\_NormalRunning, 281
- SYSTEMMODE\_RescueRunning, 281
- SYSTEMMODE\_RescueRunningFactoryReset, 281
- SYSTEMMODE\_Startup, 281
- SYSTEMMODE\_StartupRescue, 281
- SYSTEMMODE\_StartupRescueFactoryReset, 281
- SYSTEMMODE\_Unknown, 281
- TELEMATICSHANDLE, 267
- TEMP\_BACKPLANE, 277
- TEMP\_BOARD, 277
- TEMP\_BOX, 277
- TEMP\_CHIPSETS, 277
- TEMP\_CPU, 277
- TEMP\_ENV, 277
- TEMP\_OTHER, 277
- TEMP\_VIDEO, 277
- TOUCH\_NEXT\_BOOT, 281
- TOUCH\_NOW, 281
- TOUCHSCREENCALIBHANDLE, 267
- TOUCHSCREENHANDLE, 267
- TouchScreenModeSettings, 281
- TriggerConf, 281
- TS\_CALIBRATION\_MEASUREMENTS, 283
- TS\_CALIBRATION\_WIDTH, 283
- TS\_DEBOUNCE\_TIME, 283
- TS\_DEBOUNCE\_TIMEOUT\_TIME, 283
- TS\_DOUBLE\_CLICK\_TIME, 283
- TS\_DOUBLECLICK\_MAX\_CLICK\_TIME, 283
- TS\_LOW\_LEVEL, 282
- TS\_MAX\_RIGHTCLICK\_DISTANCE, 283
- TS\_RESTORE\_DEFAULT\_SETTINGS, 283
- TS\_RIGHT\_CLICK\_TIME, 282
- TS\_TCHAUTOCAL, 284
- TS\_UNTOUCHLEVEL, 283
- TS\_USE\_DEJITTER, 283
- TSAdvancedSettingsParameter, 282
- UNDEFINED\_COLOR, 271
- UPGRADE\_COMPLETE, 284
- UPGRADE\_COMPLETE\_WITH\_ERRORS, 284
- UPGRADE\_CONVERTING\_FILE, 284
- UPGRADE\_FLASHING, 284
- UPGRADE\_INIT, 284
- UPGRADE\_PREP\_COM, 284
- UPGRADE\_READING\_FILE, 284
- UPGRADE\_VERIFYING, 284
- UpgradeAction, 284
- VersionType, 267
- Video1Conf, 290
- Video2Conf, 290
- Video3Conf, 290
- Video4Conf, 290
- VideoChannel, 284
- VIDEOHANDLE, 267
- VideoRotation, 285
- videoStandard, 285
- VOLTAGE\_0V9, 286

- VOLTAGE\_12V, 285
- VOLTAGE\_12V\_BL, 286
- VOLTAGE\_12VID, 285
- VOLTAGE\_1V0, 286
- VOLTAGE\_1V05, 286
- VOLTAGE\_1V1, 286
- VOLTAGE\_1V2, 286
- VOLTAGE\_1V3\_PER, 286
- VOLTAGE\_1V3\_VDDA, 286
- VOLTAGE\_1V5, 286
- VOLTAGE\_1V8, 286
- VOLTAGE\_1V9, 286
- VOLTAGE\_24V, 285
- VOLTAGE\_24V\_BACKUP, 286
- VOLTAGE\_24VIN, 285
- VOLTAGE\_2V5, 286
- VOLTAGE\_3V3, 285
- VOLTAGE\_3V3STBY, 286
- VOLTAGE\_5V, 285
- VOLTAGE\_5V\_MP, 286
- VOLTAGE\_5VSTB, 286
- VOLTAGE\_AI\_1, 286
- VOLTAGE\_AI\_2, 286
- VOLTAGE\_AI\_3, 286
- VOLTAGE\_BL\_ANODE, 287
- VOLTAGE\_END, 287
- VOLTAGE\_IGN\_STATE, 287
- VOLTAGE\_IGN\_VALUE, 287
- VOLTAGE\_PRE\_IGN\_STATE, 287
- VOLTAGE\_PRE\_IGN\_VALUE, 287
- VOLTAGE\_UB, 286
- VOLTAGE\_VBACKUP, 287
- VOLTAGE\_VHEAT, 287
- VOLTAGE\_VIN\_FILT, 287
- VOLTAGE\_VIN\_PROT, 287
- VOLTAGE\_VMAIN, 286
- VOLTAGE\_VPMIC, 286
- VOLTAGE\_VREF\_INT, 286
- VOLTAGE\_VTFT, 286
- VoltageEnum, 285
- YELLOW, 271
- currentAction
  - UpgradeStatus, 297
- CYAN
  - CrossControl, 271
- decoder\_register
  - video\_dec\_command, 298
- DeInterlace\_BOB
  - CrossControl, 274
- DeInterlace\_Even
  - CrossControl, 274
- DeInterlace\_Odd
  - CrossControl, 274
- DeInterlaceMode
  - CrossControl, 274
- Diagnostic functions, 125
  - Diagnostic\_clearHwErrorStatus, 126
  - Diagnostic\_getHwErrorStatus, 126
  - Diagnostic\_getMinMaxTemp, 127
  - Diagnostic\_getPCBTemp, 127
  - Diagnostic\_getPMTemp, 128
  - Diagnostic\_getPowerCycles, 128
  - Diagnostic\_getShutDownReason, 129
  - Diagnostic\_getSSTemp, 129
  - Diagnostic\_getStartupReason, 130
  - Diagnostic\_getTimer, 130
  - Diagnostic\_release, 131
  - GetDiagnostic, 131
- Diagnostic\_clearHwErrorStatus
  - Diagnostic functions, 126
- Diagnostic\_getHwErrorStatus
  - Diagnostic functions, 126
- Diagnostic\_getMinMaxTemp
  - Diagnostic functions, 127
- Diagnostic\_getPCBTemp
  - Diagnostic functions, 127
- Diagnostic\_getPMTemp
  - Diagnostic functions, 128
- Diagnostic\_getPowerCycles
  - Diagnostic functions, 128
- Diagnostic\_getShutDownReason
  - Diagnostic functions, 129
- Diagnostic\_getSSTemp
  - Diagnostic functions, 129
- Diagnostic\_getStartupReason
  - Diagnostic functions, 130
- Diagnostic\_getTimer
  - Diagnostic functions, 130
- Diagnostic\_release
  - Diagnostic functions, 131
- DIAGNOSTICHANDLE
  - CrossControl, 266
- DigIO functions, 131
  - DigIO\_getDigIO, 132
  - DigIO\_getDigPowerOutput, 132
  - DigIO\_release, 133
  - DigIO\_setDigIO, 133
  - DigIO\_setDigPowerOutput, 134
  - GetDigIO, 135

- DigIO\_getDigIO
  - DigIO functions, [132](#)
- DigIO\_getDigPowerOutput
  - DigIO functions, [132](#)
- DigIO\_release
  - DigIO functions, [133](#)
- DigIO\_setDigIO
  - DigIO functions, [133](#)
- DigIO\_setDigPowerOutput
  - DigIO functions, [134](#)
- DIGIOHANDLE
  - CrossControl, [266](#)
- DigitalIn\_1
  - CrossControl, [289](#)
- DigitalIn\_2
  - CrossControl, [290](#)
- DigitalIn\_3
  - CrossControl, [290](#)
- DigitalIn\_4
  - CrossControl, [290](#)
- Disabled
  - CrossControl, [271](#)
- eErr
  - CrossControl, [274](#)
- Enabled
  - CrossControl, [271](#)
- ERR\_AVERAGE\_CALC\_STARTED
  - CrossControl, [275](#)
- ERR\_BUFFER\_SIZE
  - CrossControl, [275](#)
- ERR\_CHECKSUM
  - CrossControl, [275](#)
- ERR\_CODE\_NOT\_EXIST
  - CrossControl, [275](#)
- ERR\_COMMAND\_FAILED
  - CrossControl, [275](#)
- ERR\_CREATE\_THREAD
  - CrossControl, [275](#)
- ERR\_DATATYPE\_MISMATCH
  - CrossControl, [275](#)
- ERR\_DEVICE\_READ\_DATA\_FAILED
  - CrossControl, [275](#)
- ERR\_DEVICE\_WRITE\_DATA\_FAILED
  - CrossControl, [275](#)
- ERR\_EEPROM
  - CrossControl, [275](#)
- ERR\_END
  - CrossControl, [276](#)
- ERR\_I2C\_EXPANDER\_INIT\_FAILED
  - CrossControl, [275](#)
- ERR\_I2C\_EXPANDER\_READ\_FAILED
  - CrossControl, [275](#)
- ERR\_I2C\_EXPANDER\_WRITE\_FAILED
  - CrossControl, [275](#)
- ERR\_IN\_PROGRESS
  - CrossControl, [275](#)
- ERR\_INIT\_FAILED
  - CrossControl, [275](#)
- ERR\_INVALID\_DATA
  - CrossControl, [275](#)
- ERR\_INVALID\_PARAMETER
  - CrossControl, [275](#)
- ERR\_INVALID\_STARTUP\_TRIGGER
  - CrossControl, [276](#)
- ERR\_IOCTL\_FAILED
  - CrossControl, [275](#)
- ERR\_JIDA\_TEMP
  - CrossControl, [275](#)
- ERR\_JOIN\_THREAD
  - CrossControl, [276](#)
- ERR\_MEM\_ALLOC\_FAIL
  - CrossControl, [276](#)
- ERR\_NEWER\_FPGA\_VERSION\_REQUIRED
  - CrossControl, [275](#)
- ERR\_NEWER\_FRONT\_VERSION\_REQUIRED
  - CrossControl, [276](#)
- ERR\_NEWER\_SS\_VERSION\_REQUIRED
  - CrossControl, [275](#)
- ERR\_NOT\_RUNNING
  - CrossControl, [275](#)
- ERR\_NOT\_SUPPORTED
  - CrossControl, [274](#)
- ERR\_OPEN\_FAILED
  - CrossControl, [274](#)
- ERR\_SUCCESS
  - CrossControl, [274](#)
- ERR\_TELEMATICS\_BT\_NOT\_AVAILABLE
  - CrossControl, [276](#)
- ERR\_TELEMATICS\_GPRS\_NOT\_AVAILABLE
  - CrossControl, [276](#)
- ERR\_TELEMATICS\_GPS\_NOT\_AVAILABLE
  - CrossControl, [276](#)
- ERR\_TELEMATICS\_WLAN\_NOT\_AVAILABLE
  - CrossControl, [276](#)
- ERR\_UNKNOWN\_FEATURE
  - CrossControl, [274](#)
- ERR\_VERIFY\_FAILED
  - CrossControl, [275](#)
- errCodeNoErr

- CrossControl, 276
- errorCode
  - UpgradeStatus, 297
- ErrorStatus
  - CrossControl, 276
- ErrorStatus\_ChargeFail
  - CrossControl, 276
- ErrorStatus\_Init
  - CrossControl, 276
- ErrorStatus\_NoError
  - CrossControl, 276
- ErrorStatus\_Overcurrent
  - CrossControl, 276
- ErrorStatus\_SecondaryTempSensor
  - CrossControl, 276
- ErrorStatus\_ThermistorTempSensor
  - CrossControl, 276
- FIRMWAREUPGHANDLE
  - CrossControl, 266
- FirmwareUpgrade functions, 135
  - FirmwareUpgrade\_getUpgradeStatus, 136
  - FirmwareUpgrade\_release, 136
  - FirmwareUpgrade\_shutDown, 137
  - FirmwareUpgrade\_startDMCUUpgrade, 137
  - FirmwareUpgrade\_startDMCUVerification, 138
  - FirmwareUpgrade\_startFpgaUpgrade, 138
  - FirmwareUpgrade\_startFpgaVerification, 139
  - FirmwareUpgrade\_startFrontUpgrade, 140
  - FirmwareUpgrade\_startFrontVerification, 141
  - FirmwareUpgrade\_startSSUpgrade, 142
  - FirmwareUpgrade\_startSSVerification, 143
  - GetFirmwareUpgrade, 144
- FirmwareUpgrade\_getUpgradeStatus
  - FirmwareUpgrade functions, 136
- FirmwareUpgrade\_release
  - FirmwareUpgrade functions, 136
- FirmwareUpgrade\_shutDown
  - FirmwareUpgrade functions, 137
- FirmwareUpgrade\_startDMCUUpgrade
  - FirmwareUpgrade functions, 137
- FirmwareUpgrade\_startDMCUVerification
  - FirmwareUpgrade functions, 138
- FirmwareUpgrade\_startFpgaUpgrade
  - FirmwareUpgrade functions, 138
- FirmwareUpgrade\_startFpgaVerification
  - FirmwareUpgrade functions, 139
- FirmwareUpgrade\_startFrontUpgrade
  - FirmwareUpgrade functions, 140
- FirmwareUpgrade\_startFrontVerification
  - FirmwareUpgrade functions, 141
- FirmwareUpgrade\_startSSUpgrade
  - FirmwareUpgrade functions, 142
- FirmwareUpgrade\_startSSVerification
  - FirmwareUpgrade functions, 143
- float32\_t
  - CCAuxTypes.h, 310
- float64\_t
  - CCAuxTypes.h, 310
- FpgaLedTimingType, 293
  - idleTime, 293
  - ledNbr, 293
  - nrOfPulses, 293
  - offTime, 293
  - onTime, 293
- FrameExtended
  - CrossControl, 270
- FrameStandard
  - CrossControl, 270
- FrameStandardExtended
  - CrossControl, 270
- frequency
  - BuzzerSetup, 292
- Front\_Button\_Enabled
  - CrossControl, 282
- FrontLED functions, 145
  - FrontLED\_getBootLEDConfig, 146
  - FrontLED\_getColor, 147
  - FrontLED\_getEnabledDuringStartup, 147
  - FrontLED\_getIdleTime, 148
  - FrontLED\_getNrOfPulses, 148
  - FrontLED\_getOffTime, 148
  - FrontLED\_getOnTime, 149
  - FrontLED\_getPostBootLEDConfig, 149
  - FrontLED\_getSignal, 150
  - FrontLED\_getStandardColor, 150
  - FrontLED\_release, 151
  - FrontLED\_setBootLEDConfig, 151
  - FrontLED\_setColor, 152
  - FrontLED\_setEnabledDuringStartup, 152

- FrontLED\_setIdleTime, 153
- FrontLED\_setNrOfPulses, 153
- FrontLED\_setOff, 154
- FrontLED\_setOffTime, 154
- FrontLED\_setOnTime, 154
- FrontLED\_setPostBootLEDConfig, 155
- FrontLED\_setSignal, 156
- FrontLED\_setStandardColor, 156
- GetFrontLED, 157
- FrontLED\_getBootLEDConfig
  - FrontLED functions, 146
- FrontLED\_getColor
  - FrontLED functions, 147
- FrontLED\_getEnabledDuringStartup
  - FrontLED functions, 147
- FrontLED\_getIdleTime
  - FrontLED functions, 148
- FrontLED\_getNrOfPulses
  - FrontLED functions, 148
- FrontLED\_getOffTime
  - FrontLED functions, 148
- FrontLED\_getOnTime
  - FrontLED functions, 149
- FrontLED\_getPostBootLEDConfig
  - FrontLED functions, 149
- FrontLED\_getSignal
  - FrontLED functions, 150
- FrontLED\_getStandardColor
  - FrontLED functions, 150
- FrontLED\_release
  - FrontLED functions, 151
- FrontLED\_setBootLEDConfig
  - FrontLED functions, 151
- FrontLED\_setColor
  - FrontLED functions, 152
- FrontLED\_setEnabledDuringStartup
  - FrontLED functions, 152
- FrontLED\_setIdleTime
  - FrontLED functions, 153
- FrontLED\_setNrOfPulses
  - FrontLED functions, 153
- FrontLED\_setOff
  - FrontLED functions, 154
- FrontLED\_setOffTime
  - FrontLED functions, 154
- FrontLED\_setOnTime
  - FrontLED functions, 154
- FrontLED\_setPostBootLEDConfig
  - FrontLED functions, 155
- FrontLED\_setSignal
  - FrontLED functions, 156
- FrontLED\_setStandardColor
  - FrontLED functions, 156
- FRONTLEDHANDLE
  - CrossControl, 266
- GetAbout
  - About functions, 52
- GetAdc
  - Adc functions, 55
- GetAuxVersion
  - AuxVersion functions, 64
- GetBacklight
  - Backlight functions, 72
- GetBattery
  - Battery functions, 82
- GetBuzzer
  - Buzzer functions, 88
- GetCanSetting
  - CanSetting functions, 93
- GetCfgIn
  - CfgIn functions, 101
- GetConfig
  - Config functions, 125
- GetDiagnostic
  - Diagnostic functions, 131
- GetDigIO
  - DigIO functions, 135
- GetErrorStringA
  - CrossControl, 287
- GetErrorStringW
  - CrossControl, 288
- GetFirmwareUpgrade
  - FirmwareUpgrade functions, 144
- GetFrontLED
  - FrontLED functions, 157
- GetHwErrorStatusStringA
  - CrossControl, 288
- GetHwErrorStatusStringW
  - CrossControl, 288
- GetLightsensor
  - Lightsensor functions, 158
- GetPower
  - Power functions, 163
- GetPowerMgr
  - PowerMgr functions, 171
- GetPWMOut
  - PWMOut functions, 180
- GetSmart
  - Smart functions, 185

- GetSoftKey
  - SoftKey functions, 192
- GetStartupReasonStringA
  - CrossControl, 289
- GetStartupReasonStringW
  - CrossControl, 289
- GetTelematics
  - Telematics functions, 206
- GetTouchScreen
  - TouchScreen functions, 217
- GetTouchScreenCalib
  - TouchScreenCalib functions, 222
- GetVideo
  - Video functions, 227
- GREEN
  - CrossControl, 271
- green
  - LedColorMixType, 294
- hwErrorStatusCodes
  - CrossControl, 276
- idleTime
  - FpgaLedTimingType, 293
  - LedTimingType, 294
- IncludeFiles/About.h, 300
- IncludeFiles/Adc.h, 302
- IncludeFiles/AuxVersion.h, 302
- IncludeFiles/Backlight.h, 303
- IncludeFiles/Battery.h, 304
- IncludeFiles/Buzzer.h, 305
- IncludeFiles/CanSetting.h, 306
- IncludeFiles/CCAuxErrors.h, 307
- IncludeFiles/CCAuxTypes.h, 307
- IncludeFiles/CCPlatform.h, 311
- IncludeFiles/CfgIn.h, 311
- IncludeFiles/Config.h, 312
- IncludeFiles/Diagnostic.h, 314
- IncludeFiles/DiagnosticCodes.h, 315
- IncludeFiles/DigIO.h, 316
- IncludeFiles/FirmwareUpgrade.h, 316
- IncludeFiles/FrontLED.h, 317
- IncludeFiles/Lightsensor.h, 319
- IncludeFiles/Power.h, 319
- IncludeFiles/PowerMgr.h, 320
- IncludeFiles/PWMOut.h, 321
- IncludeFiles/Releasenotes.dox, 321
- IncludeFiles/Smart.h, 321
- IncludeFiles/SoftKey.h, 322
- IncludeFiles/Telematics.h, 323
- IncludeFiles/TouchScreen.h, 324
- IncludeFiles/TouchScreenCalib.h, 325
- IncludeFiles/Video.h, 326
- int16\_t
  - CCAuxTypes.h, 310
- int32\_t
  - CCAuxTypes.h, 311
- int64\_t
  - CCAuxTypes.h, 311
- int8\_t
  - CCAuxTypes.h, 311
- JidaSensorType
  - CrossControl, 277
- Last\_trigger\_conf
  - CrossControl, 282
- LedColorMixType, 294
  - blue, 294
  - green, 294
  - red, 294
- ledNbr
  - FpgaLedTimingType, 293
- LedTimingType, 294
  - idleTime, 294
  - nrOfPulses, 294
  - offTime, 295
  - onTime, 295
- Lightsensor functions, 157
  - GetLightsensor, 158
  - Lightsensor\_getAverageIlluminance, 158
  - Lightsensor\_getIlluminance, 159
  - Lightsensor\_getIlluminance2, 159
  - Lightsensor\_getOperatingRange, 160
  - Lightsensor\_release, 160
  - Lightsensor\_setOperatingRange, 161
  - Lightsensor\_startAverageCalc, 161
  - Lightsensor\_stopAverageCalc, 162
- Lightsensor\_getAverageIlluminance
  - Lightsensor functions, 158
- Lightsensor\_getIlluminance
  - Lightsensor functions, 159
- Lightsensor\_getIlluminance2
  - Lightsensor functions, 159
- Lightsensor\_getOperatingRange
  - Lightsensor functions, 160
- Lightsensor\_release
  - Lightsensor functions, 160
- Lightsensor\_setOperatingRange
  - Lightsensor functions, 161



- Lightsensor\_startAverageCalc
  - Lightsensor functions, [161](#)
- Lightsensor\_stopAverageCalc
  - Lightsensor functions, [162](#)
- LIGHTSENSORHANDLE
  - CrossControl, [266](#)
- LightSensorOperationRange
  - CrossControl, [277](#)
- LightSensorSamplingMode
  - CrossControl, [277](#)
- MAGENTA
  - CrossControl, [271](#)
- major
  - version\_info, [298](#)
- minor
  - version\_info, [298](#)
- MODE\_CALIBRATION\_13P
  - CrossControl, [270](#)
- MODE\_CALIBRATION\_5P
  - CrossControl, [270](#)
- MODE\_CALIBRATION\_9P
  - CrossControl, [270](#)
- MODE\_NORMAL
  - CrossControl, [270](#)
- MODE\_UNKNOWN
  - CrossControl, [270](#)
- MOUSE\_NEXT\_BOOT
  - CrossControl, [281](#)
- MOUSE\_NOW
  - CrossControl, [281](#)
- NoAction
  - CrossControl, [278](#)
- NoRequestsPending
  - CrossControl, [279](#)
- Normal
  - CrossControl, [278](#)
- nrOfPulses
  - FpgaLedTimingType, [293](#)
  - LedTimingType, [294](#)
- OCD\_OC
  - CrossControl, [278](#)
- OCD\_OK
  - CrossControl, [278](#)
- OCD\_POWER\_OFF
  - CrossControl, [278](#)
- OCDStatus
  - CrossControl, [277](#)
- offTime
  - FpgaLedTimingType, [293](#)
  - LedTimingType, [295](#)
- OnOff\_Signal\_Enabled
  - CrossControl, [282](#)
- onTime
  - FpgaLedTimingType, [293](#)
  - LedTimingType, [295](#)
- percent
  - UpgradeStatus, [297](#)
- Power functions, [162](#)
  - GetPower, [163](#)
  - Power\_ackPowerRequest, [164](#)
  - Power\_getBLPowerStatus, [164](#)
  - Power\_getButtonPowerTransitionStatus, [164](#)
  - Power\_getCanOCDSStatus, [165](#)
  - Power\_getCanPowerStatus, [166](#)
  - Power\_getExtFanPowerStatus, [166](#)
  - Power\_getVideoOCDSStatus, [167](#)
  - Power\_getVideoPowerStatus, [167](#)
  - Power\_release, [168](#)
  - Power\_setBLPowerStatus, [168](#)
  - Power\_setCanPowerStatus, [169](#)
  - Power\_setExtFanPowerStatus, [169](#)
  - Power\_setVideoPowerStatus, [170](#)
- Power\_ackPowerRequest
  - Power functions, [164](#)
- Power\_getBLPowerStatus
  - Power functions, [164](#)
- Power\_getButtonPowerTransitionStatus
  - Power functions, [164](#)
- Power\_getCanOCDSStatus
  - Power functions, [165](#)
- Power\_getCanPowerStatus
  - Power functions, [166](#)
- Power\_getExtFanPowerStatus
  - Power functions, [166](#)
- Power\_getVideoOCDSStatus
  - Power functions, [167](#)
- Power\_getVideoPowerStatus
  - Power functions, [167](#)
- Power\_release
  - Power functions, [168](#)
- Power\_setBLPowerStatus
  - Power functions, [168](#)
- Power\_setCanPowerStatus
  - Power functions, [169](#)
- Power\_setExtFanPowerStatus

- Power functions, 169
- Power\_setVideoPowerStatus
  - Power functions, 170
- PowerAction
  - CrossControl, 278
- POWERHANDLE
  - CrossControl, 267
- PowerMgr functions, 170
  - GetPowerMgr, 171
  - PowerMgr\_getConfiguration, 171
  - PowerMgr\_getPowerMgrStatus, 172
  - PowerMgr\_hasResumed, 174
  - PowerMgr\_registerControlledSuspendOrShutdown, 176
  - PowerMgr\_release, 177
  - PowerMgr\_setAppReadyForSuspendOrShutdown, 177
- PowerMgr\_getConfiguration
  - PowerMgr functions, 171
- PowerMgr\_getPowerMgrStatus
  - PowerMgr functions, 172
- PowerMgr\_hasResumed
  - PowerMgr functions, 174
- PowerMgr\_registerControlledSuspendOrShutdown
  - PowerMgr functions, 176
- PowerMgr\_release
  - PowerMgr functions, 177
- PowerMgr\_setAppReadyForSuspendOrShutdown
  - PowerMgr functions, 177
- PowerMgrConf
  - CrossControl, 278
- POWERMGRHANDLE
  - CrossControl, 267
- PowerMgrStatus
  - CrossControl, 279
- PowerOutput
  - CrossControl, 279
- PowerOutput1
  - CrossControl, 279
- PowerOutput2
  - CrossControl, 279
- PowerOutput3
  - CrossControl, 279
- PowerOutput4
  - CrossControl, 279
- PowerOutput5
  - CrossControl, 279
- PowerOutput6
  - CrossControl, 279
- PowerOutputMax
  - CrossControl, 279
- PowerSource
  - CrossControl, 279
- PowerSource\_Battery
  - CrossControl, 279
- PowerSource\_ExternalPower
  - CrossControl, 279
- PWMOut functions, 179
  - GetPWMOut, 180
  - PWMOut\_getPWMOutputChannelDutyCycle, 180
  - PWMOut\_getPWMOutputChannelFrequency, 181
  - PWMOut\_getPWMOutputStatus, 181
  - PWMOut\_release, 182
  - PWMOut\_setPWMOutOff, 182
  - PWMOut\_setPWMOutputChannelDutyCycle, 183
  - PWMOut\_setPWMOutputChannelFrequency, 184
  - PWMOut\_getPWMOutputChannelDutyCycle
    - PWMOut functions, 180
  - PWMOut\_getPWMOutputChannelFrequency
    - PWMOut functions, 181
  - PWMOut\_getPWMOutputStatus
    - PWMOut functions, 181
  - PWMOut\_release
    - PWMOut functions, 182
  - PWMOut\_setPWMOutOff
    - PWMOut functions, 182
  - PWMOut\_setPWMOutputChannelDutyCycle
    - PWMOut functions, 183
  - PWMOut\_setPWMOutputChannelFrequency
    - PWMOut functions, 184
- PWMOUTHANDLE
  - CrossControl, 267
- RangeExtended
  - CrossControl, 277
- RangeStandard
  - CrossControl, 277
- received\_framerate
  - received\_video, 295
- received\_height
  - received\_video, 295
- received\_video, 295
  - received\_framerate, 295
  - received\_height, 295
  - received\_width, 295
- received\_width

- received\_video, 295
- RED
  - CrossControl, 271
- red
  - LedColorMixType, 294
- register\_value
  - video\_dec\_command, 298
- release
  - version\_info, 298
- Rot180
  - CrossControl, 285
- Rot270
  - CrossControl, 285
- Rot90
  - CrossControl, 285
- RotNone
  - CrossControl, 285
- RS4XXPort
  - CrossControl, 279
- RS4XXPort1
  - CrossControl, 280
- RS4XXPort2
  - CrossControl, 280
- RS4XXPort3
  - CrossControl, 280
- RS4XXPort4
  - CrossControl, 280
- RunTime40\_60
  - TimerType, 296
- RunTime60\_70
  - TimerType, 296
- RunTime70\_80
  - TimerType, 296
- RunTime\_0\_40
  - BatteryTimerType, 291
- RunTime\_40\_60
  - BatteryTimerType, 291
- RunTime\_60\_70
  - BatteryTimerType, 291
- RunTime\_70\_80
  - BatteryTimerType, 291
- RunTime\_Above80
  - BatteryTimerType, 292
- RunTime\_m20
  - BatteryTimerType, 292
- RunTime\_m20\_0
  - BatteryTimerType, 292
- SamplingModeAuto
  - CrossControl, 277
- SamplingModeExtended
  - CrossControl, 277
- SamplingModeStandard
  - CrossControl, 277
- ShutdownPending
  - CrossControl, 279
- shutdownReasonCodeNoError
  - CrossControl, 280
- shutDownReasonCodes
  - CrossControl, 280
- Smart functions, 184
  - GetSmart, 185
  - Smart\_getDeviceSerial, 185
  - Smart\_getDeviceSerial2, 187
  - Smart\_getInitialTime, 188
  - Smart\_getInitialTime2, 188
  - Smart\_getRemainingLifeTime, 189
  - Smart\_getRemainingLifeTime2, 190
  - Smart\_release, 191
- Smart\_getDeviceSerial
  - Smart functions, 185
- Smart\_getDeviceSerial2
  - Smart functions, 187
- Smart\_getInitialTime
  - Smart functions, 188
- Smart\_getInitialTime2
  - Smart functions, 188
- Smart\_getRemainingLifeTime
  - Smart functions, 189
- Smart\_getRemainingLifeTime2
  - Smart functions, 190
- Smart\_release
  - Smart functions, 191
- SMARTHANDLE
  - CrossControl, 267
- SoftKey functions, 191
  - GetSoftKey, 192
  - SoftKey\_getBacklightEnabledDuringStartup, 193
  - SoftKey\_getBacklightIdleTime, 193
  - SoftKey\_getBacklightIntensity, 194
  - SoftKey\_getBacklightNrOfPulses, 194
  - SoftKey\_getBacklightOffTime, 194
  - SoftKey\_getBacklightOnTime, 195
  - SoftKey\_getBacklightSignal, 195
  - SoftKey\_getBootBacklightConfig, 196
  - SoftKey\_getMultipleBacklightIntensities, 197
  - SoftKey\_getStatus, 197
  - SoftKey\_release, 199

- SoftKey\_setBacklightEnabledDuringStartup
  - SoftKey functions, 199
- SoftKey\_setBacklightIdleTime, 200
- SoftKey\_setBacklightIntensity, 200
- SoftKey\_setBacklightNrOfPulses, 200
- SoftKey\_setBacklightOff, 201
- SoftKey\_setBacklightOffTime, 201
- SoftKey\_setBacklightOnTime, 202
- SoftKey\_setBacklightSignal, 202
- SoftKey\_setBootBacklightConfig, 203
- SoftKey\_setMultipleBacklightIntensities
  - SoftKey functions, 204
- SoftKey\_getBacklightEnabledDuringStartup
  - SoftKey functions, 193
- SoftKey\_getBacklightIdleTime
  - SoftKey functions, 193
- SoftKey\_getBacklightIntensity
  - SoftKey functions, 194
- SoftKey\_getBacklightNrOfPulses
  - SoftKey functions, 194
- SoftKey\_getBacklightOffTime
  - SoftKey functions, 194
- SoftKey\_getBacklightOnTime
  - SoftKey functions, 195
- SoftKey\_getBacklightSignal
  - SoftKey functions, 195
- SoftKey\_getBootBacklightConfig
  - SoftKey functions, 196
- SoftKey\_getMultipleBacklightIntensities
  - SoftKey functions, 197
- SoftKey\_getStatus
  - SoftKey functions, 197
- SoftKey\_release
  - SoftKey functions, 199
- SoftKey\_setBacklightEnabledDuringStartup
  - SoftKey functions, 199
- SoftKey\_setBacklightIdleTime
  - SoftKey functions, 200
- SoftKey\_setBacklightIntensity
  - SoftKey functions, 200
- SoftKey\_setBacklightNrOfPulses
  - SoftKey functions, 200
- SoftKey\_setBacklightOff
  - SoftKey functions, 201
- SoftKey\_setBacklightOffTime
  - SoftKey functions, 201
- SoftKey\_setBacklightOnTime
  - SoftKey functions, 202
- SoftKey\_setBacklightSignal
  - SoftKey functions, 202
- SoftKey\_setBootBacklightConfig
  - SoftKey functions, 203
- SoftKey\_setMultipleBacklightIntensities
  - SoftKey functions, 204
- SOFTKEYHANDLE
  - CrossControl, 267
- startupReasonAlwaysStart
  - CrossControl, 280
- startupReasonCodeButtonPress
  - CrossControl, 280
- startupReasonCodeCanActivity
  - CrossControl, 280
- startupReasonCodeCIAActivity
  - CrossControl, 280
- startupReasonCodeExtCtrl
  - CrossControl, 280
- startupReasonCodeMPRestart
  - CrossControl, 280
- startupReasonCodePowerOnStartup
  - CrossControl, 280
- startupReasonCodes
  - CrossControl, 280
- startupReasonCodeUndefined
  - CrossControl, 280
- startupReasonUnknownTrigger
  - CrossControl, 280
- STD\_B\_D\_G\_H\_I\_N\_PAL
  - CrossControl, 285
- STD\_M\_J\_NTSC
  - CrossControl, 285
- STD\_M\_PAL
  - CrossControl, 285
- STD\_NTSC
  - CrossControl, 285
- STD\_PAL
  - CrossControl, 285
- STD\_SECAM
  - CrossControl, 285
- SuspendPending
  - CrossControl, 279
- SystemMode
  - CrossControl, 280
- SYSTEMMODE\_NormalRunning
  - CrossControl, 281
- SYSTEMMODE\_RescueRunning
  - CrossControl, 281
- SYSTEMMODE\_RescueRunningFactoryReset
  - CrossControl, 281
- SYSTEMMODE\_Startup
  - CrossControl, 281

- SYSTEMMODE\_StartupRescue
  - CrossControl, 281
- SYSTEMMODE\_StartupRescueFactoryReset
  - CrossControl, 281
- SYSTEMMODE\_Unknown
  - CrossControl, 281
- Telematics functions, 205
  - GetTelematics, 206
  - Telematics\_getBTPowerStatus, 206
  - Telematics\_getBTStartUpPowerStatus, 207
  - Telematics\_getGPRSPowerStatus, 207
  - Telematics\_getGPRSStartUpPowerStatus, 208
  - Telematics\_getGPSAntennaStatus, 209
  - Telematics\_getGPSPowerStatus, 209
  - Telematics\_getGPSStartUpPowerStatus, 210
  - Telematics\_getTelematicsAvailable, 211
  - Telematics\_getWLANPowerStatus, 211
  - Telematics\_getWLANStartUpPowerStatus, 212
  - Telematics\_release, 212
  - Telematics\_setBTPowerStatus, 213
  - Telematics\_setBTStartUpPowerStatus, 213
  - Telematics\_setGPRSPowerStatus, 214
  - Telematics\_setGPRSStartUpPowerStatus, 214
  - Telematics\_setGPSPowerStatus, 214
  - Telematics\_setGPSStartUpPowerStatus, 215
  - Telematics\_setWLANPowerStatus, 215
  - Telematics\_setWLANStartUpPowerStatus, 216
- Telematics\_getBTPowerStatus
  - Telematics functions, 206
- Telematics\_getBTStartUpPowerStatus
  - Telematics functions, 207
- Telematics\_getGPRSPowerStatus
  - Telematics functions, 207
- Telematics\_getGPRSStartUpPowerStatus
  - Telematics functions, 208
- Telematics\_getGPSAntennaStatus
  - Telematics functions, 209
- Telematics\_getGPSPowerStatus
  - Telematics functions, 209
- Telematics\_getGPSStartUpPowerStatus
  - Telematics functions, 210
- Telematics\_getTelematicsAvailable
  - Telematics functions, 211
- Telematics\_getWLANPowerStatus
  - Telematics functions, 211
- Telematics\_getWLANStartUpPowerStatus
  - Telematics functions, 212
- Telematics\_release
  - Telematics functions, 212
- Telematics\_setBTPowerStatus
  - Telematics functions, 213
- Telematics\_setBTStartUpPowerStatus
  - Telematics functions, 213
- Telematics\_setGPRSPowerStatus
  - Telematics functions, 214
- Telematics\_setGPRSStartUpPowerStatus
  - Telematics functions, 214
- Telematics\_setGPSPowerStatus
  - Telematics functions, 214
- Telematics\_setGPSStartUpPowerStatus
  - Telematics functions, 215
- Telematics\_setWLANPowerStatus
  - Telematics functions, 215
- Telematics\_setWLANStartUpPowerStatus
  - Telematics functions, 216
- TELEMATICSHANDLE
  - CrossControl, 267
- TEMP\_BACKPLANE
  - CrossControl, 277
- TEMP\_BOARD
  - CrossControl, 277
- TEMP\_BOX
  - CrossControl, 277
- TEMP\_CHIPSETS
  - CrossControl, 277
- TEMP\_CPU
  - CrossControl, 277
- TEMP\_ENV
  - CrossControl, 277
- TEMP\_OTHER
  - CrossControl, 277
- TEMP\_VIDEO
  - CrossControl, 277
- TimerType, 296
  - Above80RunTime, 296
  - RunTime40\_60, 296
  - RunTime60\_70, 296
  - RunTime70\_80, 296
  - TotHeatTime, 296
  - TotRunTime, 296
  - TotSuspTime, 296

- TotHeatTime
  - TimerType, 296
- TotRunTime
  - TimerType, 296
- TotRunTimeBattery
  - BatteryTimerType, 292
- TotRunTimeMain
  - BatteryTimerType, 292
- TotSuspTime
  - TimerType, 296
- TOUCH\_NEXT\_BOOT
  - CrossControl, 281
- TOUCH\_NOW
  - CrossControl, 281
- TouchScreen functions, 216
  - GetTouchScreen, 217
  - TouchScreen\_getAdvancedSetting, 217
  - TouchScreen\_getMode, 218
  - TouchScreen\_getMouseRightClickTime, 218
  - TouchScreen\_release, 219
  - TouchScreen\_setAdvancedSetting, 219
  - TouchScreen\_setMode, 221
  - TouchScreen\_setMouseRightClickTimeTriggerConf, 221
- TouchScreen\_getAdvancedSetting
  - TouchScreen functions, 217
- TouchScreen\_getMode
  - TouchScreen functions, 218
- TouchScreen\_getMouseRightClickTime
  - TouchScreen functions, 218
- TouchScreen\_release
  - TouchScreen functions, 219
- TouchScreen\_setAdvancedSetting
  - TouchScreen functions, 219
- TouchScreen\_setMode
  - TouchScreen functions, 221
- TouchScreen\_setMouseRightClickTime
  - TouchScreen functions, 221
- TouchScreenCalib functions, 222
  - GetTouchScreenCalib, 222
  - TouchScreenCalib\_autoSensorCalib, 223
  - TouchScreenCalib\_checkCalibrationPointFinished, 223
  - TouchScreenCalib\_getConfigParam, 223
  - TouchScreenCalib\_getMode, 224
  - TouchScreenCalib\_release, 224
  - TouchScreenCalib\_setCalibrationPoint, 224
  - TouchScreenCalib\_setConfigParam, 225
  - TouchScreenCalib\_setMode, 225
  - TouchScreenCalib\_autoSensorCalib
    - TouchScreenCalib functions, 223
  - TouchScreenCalib\_checkCalibrationPointFinished
    - TouchScreenCalib functions, 223
  - TouchScreenCalib\_getConfigParam
    - TouchScreenCalib functions, 223
  - TouchScreenCalib\_getMode
    - TouchScreenCalib functions, 224
  - TouchScreenCalib\_release
    - TouchScreenCalib functions, 224
  - TouchScreenCalib\_setCalibrationPoint
    - TouchScreenCalib functions, 224
  - TouchScreenCalib\_setConfigParam
    - TouchScreenCalib functions, 225
  - TouchScreenCalib\_setMode
    - TouchScreenCalib functions, 225
- TOUCHSCREENCALIBHANDLE
  - CrossControl, 267
- TOUCHSCREENHANDLE
  - CrossControl, 267
- TouchScreenModeSettings
  - CrossControl, 281
- TouchScreen\_setMouseRightClickTimeTriggerConf
  - CrossControl, 281
- TS\_CALIBRATION\_MEASUREMENTS
  - CrossControl, 283
- TS\_CALIBRATION\_WIDTH
  - CrossControl, 283
- TS\_DEBOUNCE\_TIME
  - CrossControl, 283
- TS\_DEBOUNCE\_TIMEOUT\_TIME
  - CrossControl, 283
- TS\_DOUBLE\_CLICK\_TIME
  - CrossControl, 283
- TS\_DOUBLECLICK\_MAX\_CLICK\_TIME
  - CrossControl, 283
- TS\_LOW\_LEVEL
  - CrossControl, 282
- TS\_MAX\_RIGHTCLICK\_DISTANCE
  - CrossControl, 283
- TS\_RESTORE\_DEFAULT\_SETTINGS
  - CrossControl, 283
- TS\_RIGHT\_CLICK\_TIME
  - CrossControl, 282
- TS\_TCHAUTOCAL
  - CrossControl, 284
- TS\_UNTOUCHLEVEL
  - CrossControl, 283
- TS\_USE\_DEJITTER

- CrossControl, 283
- TSAdvancedSettingsParameter
  - CrossControl, 282
- uint16\_t
  - CCAuxTypes.h, 311
- uint32\_t
  - CCAuxTypes.h, 311
- uint64\_t
  - CCAuxTypes.h, 311
- uint8\_t
  - CCAuxTypes.h, 311
- UNDEFINED\_COLOR
  - CrossControl, 271
- UPGRADE\_COMPLETE
  - CrossControl, 284
- UPGRADE\_COMPLETE\_WITH\_ERRORS
  - CrossControl, 284
- UPGRADE\_CONVERTING\_FILE
  - CrossControl, 284
- UPGRADE\_FLASHING
  - CrossControl, 284
- UPGRADE\_INIT
  - CrossControl, 284
- UPGRADE\_PREP\_COM
  - CrossControl, 284
- UPGRADE\_READING\_FILE
  - CrossControl, 284
- UPGRADE\_VERIFYING
  - CrossControl, 284
- UpgradeAction
  - CrossControl, 284
- UpgradeStatus, 297
  - currentAction, 297
  - errorCode, 297
  - percent, 297
- version\_info, 297
  - build, 298
  - major, 298
  - minor, 298
  - release, 298
- VersionType
  - CrossControl, 267
- Video functions, 226
  - GetVideo, 227
  - Video\_activateSnapshot, 228
  - Video\_createBitmap, 228
  - Video\_freeBmpBuffer, 229
  - Video\_getActiveChannel, 229
  - Video\_getColorKeys, 230
  - Video\_getCropping, 230
  - Video\_getDecoderReg, 231
  - Video\_getDeInterlaceMode, 231
  - Video\_getGraphicsOverlay, 232
  - Video\_getMirroring, 232
  - Video\_getRawImage, 233
  - Video\_getRotation, 233
  - Video\_getScaling, 233
  - Video\_getStatus, 234
  - Video\_getVideoArea, 234
  - Video\_getVideoStandard, 235
  - Video\_init, 235
  - Video\_minimize, 236
  - Video\_release, 236
  - Video\_restore, 236
  - Video\_setActiveChannel, 237
  - Video\_setColorKeys, 237
  - Video\_setCropping, 238
  - Video\_setDecoderReg, 238
  - Video\_setDeInterlaceMode, 239
  - Video\_setGraphicsOverlay, 239
  - Video\_setMirroring, 240
  - Video\_setRotation, 240
  - Video\_setScaling, 241
  - Video\_setVideoArea, 241
  - Video\_showFrame, 242
  - Video\_showVideo, 242
  - Video\_takeSnapshot, 242
  - Video\_takeSnapshotBmp, 243
  - Video\_takeSnapshotRaw, 243
- Video1Conf
  - CrossControl, 290
- Video2Conf
  - CrossControl, 290
- Video3Conf
  - CrossControl, 290
- Video4Conf
  - CrossControl, 290
- Video\_activateSnapshot
  - Video functions, 228
- Video\_createBitmap
  - Video functions, 228
- video\_dec\_command, 298
  - decoder\_register, 298
  - register\_value, 298
- Video\_freeBmpBuffer
  - Video functions, 229
- Video\_getActiveChannel
  - Video functions, 229

- Video\_getColorKeys
  - Video functions, 230
- Video\_getCropping
  - Video functions, 230
- Video\_getDecoderReg
  - Video functions, 231
- Video\_getDeInterlaceMode
  - Video functions, 231
- Video\_getGraphicsOverlay
  - Video functions, 232
- Video\_getMirroring
  - Video functions, 232
- Video\_getRawImage
  - Video functions, 233
- Video\_getRotation
  - Video functions, 233
- Video\_getScaling
  - Video functions, 233
- Video\_getStatus
  - Video functions, 234
- Video\_getVideoArea
  - Video functions, 234
- Video\_getVideoStandard
  - Video functions, 235
- Video\_init
  - Video functions, 235
- Video\_minimize
  - Video functions, 236
- Video\_release
  - Video functions, 236
- Video\_restore
  - Video functions, 236
- Video\_setActiveChannel
  - Video functions, 237
- Video\_setColorKeys
  - Video functions, 237
- Video\_setCropping
  - Video functions, 238
- Video\_setDecoderReg
  - Video functions, 238
- Video\_setDeInterlaceMode
  - Video functions, 239
- Video\_setGraphicsOverlay
  - Video functions, 239
- Video\_setMirroring
  - Video functions, 240
- Video\_setRotation
  - Video functions, 240
- Video\_setScaling
  - Video functions, 241
- Video\_setVideoArea
  - Video functions, 241
- Video\_showFrame
  - Video functions, 242
- Video\_showVideo
  - Video functions, 242
- Video\_takeSnapshot
  - Video functions, 242
- Video\_takeSnapshotBmp
  - Video functions, 243
- Video\_takeSnapshotRaw
  - Video functions, 243
- VideoChannel
  - CrossControl, 284
- VIDEOHANDLE
  - CrossControl, 267
- VideoRotation
  - CrossControl, 285
- videoStandard
  - CrossControl, 285
- VOLTAGE\_0V9
  - CrossControl, 286
- VOLTAGE\_12V
  - CrossControl, 285
- VOLTAGE\_12V\_BL
  - CrossControl, 286
- VOLTAGE\_12VID
  - CrossControl, 285
- VOLTAGE\_1V0
  - CrossControl, 286
- VOLTAGE\_1V05
  - CrossControl, 286
- VOLTAGE\_1V1
  - CrossControl, 286
- VOLTAGE\_1V2
  - CrossControl, 286
- VOLTAGE\_1V3\_PER
  - CrossControl, 286
- VOLTAGE\_1V3\_VDDA
  - CrossControl, 286
- VOLTAGE\_1V5
  - CrossControl, 286
- VOLTAGE\_1V8
  - CrossControl, 286
- VOLTAGE\_1V9
  - CrossControl, 286
- VOLTAGE\_24V
  - CrossControl, 285
- VOLTAGE\_24V\_BACKUP
  - CrossControl, 286



- VOLTAGE\_24VIN
  - CrossControl, [285](#)
- VOLTAGE\_2V5
  - CrossControl, [286](#)
- VOLTAGE\_3V3
  - CrossControl, [285](#)
- VOLTAGE\_3V3STBY
  - CrossControl, [286](#)
- VOLTAGE\_5V
  - CrossControl, [285](#)
- VOLTAGE\_5V\_MP
  - CrossControl, [286](#)
- VOLTAGE\_5VSTB
  - CrossControl, [286](#)
- VOLTAGE\_AI.1
  - CrossControl, [286](#)
- VOLTAGE\_AI.2
  - CrossControl, [286](#)
- VOLTAGE\_AI.3
  - CrossControl, [286](#)
- VOLTAGE\_BL\_ANODE
  - CrossControl, [287](#)
- VOLTAGE\_END
  - CrossControl, [287](#)
- VOLTAGE\_IGN.STATE
  - CrossControl, [287](#)
- VOLTAGE\_IGN.VALUE
  - CrossControl, [287](#)
- VOLTAGE\_PRE\_IGN.STATE
  - CrossControl, [287](#)
- VOLTAGE\_PRE\_IGN.VALUE
  - CrossControl, [287](#)
- VOLTAGE\_UB
  - CrossControl, [286](#)
- VOLTAGE\_VBACKUP
  - CrossControl, [287](#)
- VOLTAGE\_VHEAT
  - CrossControl, [287](#)
- VOLTAGE\_VIN\_FILT
  - CrossControl, [287](#)
- VOLTAGE\_VIN\_PROT
  - CrossControl, [287](#)
- VOLTAGE\_VMAIN
  - CrossControl, [286](#)
- VOLTAGE\_VPMIC
  - CrossControl, [286](#)
- VOLTAGE\_VREF\_INT
  - CrossControl, [286](#)
- VOLTAGE\_VTFT
  - CrossControl, [286](#)
- VoltageEnum
  - CrossControl, [285](#)
- volume
  - BuzzerSetup, [292](#)
- YELLOW
  - CrossControl, [271](#)