CCAux

2.15.2.0

Wed Jul 08 2020

Contents

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, XM, VI2 and VS products in all variations. For the XM and XL platforms, Windows XP, Windows 7 and 8 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

1.2 Changelog

1.2.1 Version 2.15.2.0 - VC/VA platform

- Bugfix for Video_getDecoderReg which was broken in v2.9.0.0.

1.2.2 Version 2.15.1.0 - VC/VA platform

- Bugfix for a multithreading issue affecting the following classes:
  - About

  - Auxversion
  - Backlight
  - Battery
  - Buzzer
  - CfgIn
  - Config
  - Diagnostic
  - FrontLED
  - Telematics

1.2.3 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

1.2.4 Version 2.14.0.0 - all platforms

- New function 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

1.2.5 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

1.2.6 Version 2.12.0.0 - VS platform

- New functions Buzzer_get/setScaledVolume for all platforms. Replaces Buzzer_get/setVolume which may be removed in a future version.
- CrossControl Software License Agreement replaces the Maximatecc Software License Agreement.

1.2.7 Version 2.11.0.0 - VS platform

- New functions Config_get/setShortBeepSettings, Config_get/setLongBeepSettings, FrontLED_get/setBootLEDConfig and FrontLED_get/setPostBootLEDConfig for VS platform.
- About_getNrOfUSBConnections now reports correct number of USB ports for VS platform.
- About_hasOsBooted support added for VS platform.

1.2.8 Version 2.10.0.0 - VS platform

- Initial support for VS platform.
- New functions DigIO_get/setDigPowerOutput and Config_get/setDigPowerOutput↩ StartupConfig for VS platform.

1.2.9 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.

1.2.10 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_[set|get]OnOffSignalState API functions added.
- Config_[set|get]KeySwitchTriggerMode 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.

1.2.11 Version 2.7.4.0 - VC Linux platform

- Bugfixes and documentation updates.

1.2.12 Version 2.7.3.0 - VC Linux platform

- Added functions CfgIn_getMinFrequencyThreshold and CfgIn_setMinFrequency↩ Threshold.
- Added restrictions on the usage of start-up triggers in combination with button configurations.

1.2.13 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.

1.2.14 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.

1.2.15 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: -

1.2.16 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

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

- XA/XS: Functions added: Video_getGraphicsOverlay and Video_setGraphics↩ Overlay.
- 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

1.2.18 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_get↩ InitialTime2).
- 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_SUPP↩ ORTED:
  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.

1.2.19 Version 2.4.7.0 - XM Linux platform

- XM: Improved fault-handling in function registerControlledSuspendOrShut↩ Down()

- Known issues:
  - Same as 2.4.6.0 release

1.2.20 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)

1.2.21 Version 2.4.2.0 - XA/XS platform

- XA/XS: Config_get/setRS485Mode now uses settings file for intermediate storage

- Known issues:
  - Same as 2.4.0.0 release

1.2.22 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 Lightsensor
- 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 Smart 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 compability. 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

1.2.23 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

1.2.24 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

1.2.25 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

1.2.26 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

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 CrossControl Namespace Reference

Data Structures

- struct BatteryTimerType
- struct BuzzerSetup
- struct FpgaLedTimingType
- struct LedColorMixType
- struct LedTimingType
- struct received_video
- struct TimerType
- struct UpgradeStatus
- struct version_info
- struct video_dec_command

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_AI_1, VOLTAGE_AI_2, VOLTAGE_AI_3, 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_IOCTRL_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_FAILE = 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 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_TR
  = 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_TRIGG
  }
- enum PowerOutput {
  PowerOutput1 = 0, PowerOutput2, PowerOutput3, PowerOutput4,
  PowerOutput5, PowerOutput6, PowerOutputMax }
- enum SystemMode {
  SYSTEMMODE_Startup = 0, SYSTEMMODE_StartupRescue = 1, SYSTEMMODE_StartupRescueFactoryRese
  = 2, SYSTEMMODE_NormalRunning = 3,
  SYSTEMMODE_RescueRunning = 4, SYSTEMMODE_RescueRunningFactoryReset
  = 5, SYSTEMMODE_Unknown = 6 }
- enum startupReasonCodes {
  startupReasonCodeUndefined = 0x0000, startupReasonCodeButtonPress = 0x0055,
  startupReasonCodeExtCtrl = 0x00AA, startupReasonCodeMPRestart = 0x00↩
  F0,
  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_CO↩ NV 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_CA↩ LLING_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 BACKLIGHTHANDLE CCAUXDLL_CAL↩ LING_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_CALLIN↩ G_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_CAL↩ LING_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 char_t const *CCAUXDLL_CALLING_CONV GetErrorStringA (eErr errCode)
- EXTERN_C CCAUXDLL_API wchar_t const *CCAUXDLL_CALLING_CO↩ NV 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_CAL↩ LING_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_CO↩ NV 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_CO↩ NV GetStartupReasonStringW (uint16_t code)
- EXTERN_C CCAUXDLL_API DIGIOHANDLE CCAUXDLL_CALLING_C↩ ONV 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_C↩ ALLING_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_getUpgradeStatus (FIRMWAREUPGHANDLE, UpgradeStatus *status, bool blocking)
- EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV FirmwareUpgrade_shutDown (FIRMWAREUPGHANDLE)
- EXTERN_C CCAUXDLL_API FRONTLEDHANDLE CCAUXDLL_CALLI↩ NG_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_CAL↩ LING_CONV GetPowerMgr (void)
- EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV PowerMgr_release (POWERMGRHANDLE)
- EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV PowerMgr_registerControlledSuspendOr
(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)
- EXTERN_C CCAUXDLL_API PWMOUTHANDLE CCAUXDLL_CALLIN↩ G_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)
- 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_CALLIN↩ G_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_getBacklightEnabledDuringStar (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)
- EXTERN_C CCAUXDLL_API TELEMATICSHANDLE CCAUXDLL_CAL↩ LING_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_getWLANStartUpPowerStatu (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_setWLANStartUpPowerStatu (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_C↩ ALLING_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 CCAUX↩ DLL_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 *bottom↩ RigthX, 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 b↩ Interlaced)

- 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)

5.1.1 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
```

5.1.2 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. Plaform 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

Enumerator

BUTTON_AS_BACKLIGHT_DECR↩ EASE	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_INCR↩ EASE	Set button to act as backlight decreaste 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

BPTS_No_Change	No change
BPTS_ShutDown	A shutdown has been initiated since the front panel button has been pressed longer than the set FrontBtnShutDownTrigTime
BPTS_Suspend	Suspend mode has been initiated since the front panel button has been pressed (shortly) and suspend mode is enabled
BPTS_Restart	Not currently in use
BPTS_BtnPressed	The front panel button is currently pressed. It has not been released and it has not yet been held longer than FrontBtnShutDownTrigTime.
BPTS_BtnPressedLong	The front panel button is currently pressed. It has not been released and it has been held longer than FrontBtnShutDownTrigTime.
BPTS_SignalOff	The external on/off signal is low, but not yet long enough for the ExtOnOffSigSuspTrigTime.
BPTS_END	Mark end of BPTS - list

CalibrationConfigParam

`enum CalibrationConfigParam`

Touch screen calibration parameters

Enumerator

CONFIG_CALIBRATION_WITH	
CONFIG_CALIBRATION_MEASUREMENTS	Accepted error value when calibrating.
CONFIG_5P_CALIBRATION_POINT_BORDER	Number of measurements to accept a calibration point.
CONFIG_13P_CALIBRATION_POINT_BORDER	The number of pixels from the border where the 5 point calibration points should be located.
CONFIG_13P_CALIBRATION_TRANSITION_MIN	The number of pixels from the border where the 13 point calibration points should be located.
CONFIG_13P_CALIBRATION_TRANSITION_MAX	Min defines the transition area in number of pixels, where the two different calibrations are used.

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	

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 CfgIn_getValue
CFGIN_VOLTAGE_2V5	Read digital input value with CfgIn_getValue
CFGIN_VOLTAGE_5V	Read voltage input value with CfgIn_getValue, 2.5V range
CFGIN_RESISTANCE	Read voltage input value with CfgIn_getValue, 5V range
CFGIN_FREQ_FLOATING	Read resistance input value with CfgIn_getValue, ports 1-4 only
CFGIN_FREQ_PULLUP	Read frequency value with CfgIn_getPwmValue

Enumerator

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
CFGIN_FREQ_F_10V	Read frequency value with CfgIn_getFrequencyValue, floating, 5V range, 2.5V threshold, VA only

Enumerator

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 chareged
ChargingStatus_TempLow	The temperature is too low to allow the battery to be charged
ChargingStatus_TempHigh	The temperature is too high to allow the battery to be charged
ChargingStatus_Unknown	There was an error determining the charging status

ConfigOnOffTriggerMode

enum ConfigOnOffTriggerMode

Enumeration of OnOff/Ignition/KeySwitch signal trigger modes

Enumerator

CONFIG_ONOFF_EDGE_TRIGGER	OnOff/Ignition/KeySwitch signal trigger on signal edge (default)
CONFIG_ONOFF_LEVEL_TRIGGER	OnOff/Ignition/KeySwitch signal trigger on signal level

DeInterlaceMode

enum DeInterlaceMode

The available deinterlace modes

Enumerator

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

eErr

enum eErr

Error code enumeration

Enumerator

ERR_SUCCESS	Success
ERR_OPEN_FAILED	Open failed
ERR_NOT_SUPPORTED	Not supported
ERR_UNKNOWN_FEATURE	Unknown feature
ERR_DATATYPE_MISMATCH	Datatype mismatch
ERR_CODE_NOT_EXIST	Code doesn't exist
ERR_BUFFER_SIZE	Buffer size error
ERR_IOCTRL_FAILED	IoCtrl operation failed
ERR_INVALID_DATA	Invalid data
ERR_INVALID_PARAMETER	Invalid parameter
ERR_CREATE_THREAD	Invalid parameter

Enumerator

ERR_IN_PROGRESS	Failed to create thread
ERR_CHECKSUM	Operation in progress
ERR_INIT_FAILED	Checksum error
ERR_VERIFY_FAILED	Initalization failed
ERR_DEVICE_READ_DATA_FAIL↩ ED	Failed to verify
ERR_DEVICE_WRITE_DATA_FAI↩ LED	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_FAI↩ LED	Thread isn't running
ERR_I2C_EXPANDER_WRITE_FA↩ ILED	I2C read failure
ERR_I2C_EXPANDER_INIT_FAILED	I2C write failure
ERR_NEWER_SS_VERSION_REQ↩ UIRED	I2C initialization failure
ERR_NEWER_FPGA_VERSION_R↩ EQUIRED	SS version too old
ERR_NEWER_FRONT_VERSION_↩ REQUIRED	FPGA version too old
ERR_TELEMATICS_GPRS_NOT_A↩ VAILABLE	FRONT version too old
ERR_TELEMATICS_WLAN_NOT_↩ AVAILABLE	GPRS module not available
ERR_TELEMATICS_BT_NOT_AV↩ AILABLE	WLAN module not available
ERR_TELEMATICS_GPS_NOT_AV↩ AILABLE	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

OCD_OK	Normal operation, no overcurrent condition detected
OCD_OC	Overcurrent has been detected, power has therefore been turned off, but may be functioning again if the problem disappeared after re-test
OCD_POWER_OFF	Overcurrent has been detected, power has been permanently turned off

PowerAction

`enum PowerAction`

Button and on/off signal actions.

Enumerator

NoAction	No action taken

Enumerator

ActionSuspend	The system enters suspend mode
ActionShutDown	The system shuts down

PowerMgrConf

`enum PowerMgrConf`

Enumeration of the settings that can be used with the PowerMgr system.

Enumerator

Normal	Applications will not be able to delay suspend/shutdown requests. This is the normal configuration that is used when the PowerMgr class is not being used. Setting this configuration turns off the feature if it is in use.
ApplicationControlled	Applications can delay suspend/shutdown requests.
BatterySuspend	In this mode, the computer will automatically enter suspend mode when the unit starts running on battery power. Applications can delay suspend/shutdown requests. This mode is only applicable if the unit has an external battery. Using this configuration on a computer without an external battery will be the same as using the configuration ApplicationControlled.

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	

Enumerator

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

startupReasonCodeUndefined	
startupReasonCodeButtonPress	Unknown startup reason.
startupReasonCodeExtCtrl	The system was started by front panel button press
startupReasonCodeMPRestart	The system was started by the external control signal
startupReasonCodePowerOnStartup	The system was restarted by OS request
startupReasonCodeCanActivity	The system was started due to the PowerOnStartup setting
startupReasonCodeCIActivity	The system was started due to activity on the Can bus (CCpilot VC family)
startupReasonAlwaysStart	The system was started due to activity on the configurable input signals (CCpilot VC family)
startupReasonUnknownTrigger	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 (Configurabe 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

Enumerator

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.
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.

Enumerator

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_MEASUREME↩ NTS	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.
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

Enumerator

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

Enumerator

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
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_END	< V AI_3 VI2 < Marks end of voltages - list

5.1.3 Function Documentation

About_getAddOnHWversion()

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

Get Add on hardware version.
Supported Platform(s): XL, XM, XS, XA

Parameters

buff	Text output buffer.
len	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getAdd↩
OnManufacturingDate (
            ABOUTHANDLE ,
            char_t * buff,
            int32_t len )
```

Get Add on manufacturing date.
Supported Platform(s): XL, XM, XS, XA

Parameters

buff	Text output buffer.
len	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getAdd↩
OnPCBArt (
            ABOUTHANDLE ,
            char_t * buff,
            int32_t length )
```

Get Add on PCB article number.
Supported Platform(s): XL, XM, XS, XA

Parameters

buff	Text output buffer.
length	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getAdd↩
OnPCBSerial (
            ABOUTHANDLE ,
            char_t * buff,
            int32_t len )
```

Get Add on PCB serial number.
Supported Platform(s): XL, XM, XS, XA

Parameters

buff	Text output buffer.
len	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getDisplay↩
Resolution (
            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

buff	Text output buffer.
len	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getFront↩
PcbRev (
```

```
            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

major	The major pcb revision.
minor	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_getI↩
OExpanderValue (
            ABOUTHANDLE ,
            uint16_t * value )
```

Get Value for IO Expander
Supported Platform(s): XA, XS

Parameters

value	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

mounted	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_getIs↩
BTMounted (
            ABOUTHANDLE ,
            bool * mounted )
```

Get BlueTooth module mounting status.
Supported Platform(s): XL, XM, XS, XA

Parameters

mounted	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_getIs↩
DisplayAvailable (
            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

available	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_getIs↩
GPRSMounted (
            ABOUTHANDLE ,
            bool * mounted )
```

Get GPRS module mounting status.
Supported Platform(s): XL, XM, XS, XA

Parameters

mounted	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_getIs↩
GPSMounted (
            ABOUTHANDLE ,
            bool * mounted )
```

Get GPS module mounting status.
Supported Platform(s): XL, XM, XS, XA

Parameters

mounted	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_getIs↩
IOExpanderMounted (
            ABOUTHANDLE ,
            bool * mounted )
```

Get IO Expander mounting status.
Supported Platform(s): XA, XS

Parameters

mounted	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_getIs↩
PCBBluetoothAvailable (
            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

available	Is bluetooth available?

Returns

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

Example Usage:

About_getIsTouchScreenAvailable()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getIs↩
TouchScreenAvailable (
            ABOUTHANDLE ,
            bool * available )
```

Get Display TouchScreen status.
Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

available	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_getIs↩
WLANMounted (
            ABOUTHANDLE ,
            bool * mounted )
```

Get WLAN module mounting status.
Supported Platform(s): XL, XM, XS, XA

Parameters

mounted	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_getMain↩
HWversion (
              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

buff	Text output buffer.
len	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_getMain↩
ManufacturingDate (
              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

buff	Text output buffer.
len	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_getMain↩
PCBArt (
            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

buff	Text output buffer.
length	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getMain↩
PCBSerial (
```

```
            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

buff	Text output buffer.
len	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getMain↩
ProdArtNr (
            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

buff	Text output buffer.
len	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_getMain↩
ProdRev (
            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

buff	Text output buffer.
len	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_getNr↩
OfAnalogInputs (
            ABOUTHANDLE ,
            int32_t * numanalogins )
```

Get number of external analog inputs.
Supported Platform(s): VI2

Parameters

numanalogins	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_getNr↩
OfButtons (
            ABOUTHANDLE ,
            int32_t * numbuttons )
```

Get number of configurable buttons.
Supported Platform(s): VC, VA, VS, VI2

Parameters

numbuttons	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_getNr↩
OfCANConnections (
            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

NrOfConnections	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_getNr↩
OfCfgInConnections (
            ABOUTHANDLE ,
            uint8_t * NrOfConnections )
```

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

Parameters

NrOfConnections	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_getNr↩
OfDigIOConnections (
            ABOUTHANDLE ,
            uint8_t * NrOfConnections )
```

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

Parameters

NrOfConnections	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNr↩
OfETHConnections (
            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

NrOfConnections	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_getNr↩
OfPWMOutConnections (
            ABOUTHANDLE ,
            uint8_t * NrOfConnections )
```

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

Parameters

NrOfConnections	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_getNr↩
OfSerialConnections (
            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

NrOfConnections	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_getNr↩
OfUSBConnections (
            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

NrOfConnections	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_getNr↩
OfVideoConnections (
            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

NrOfConnections	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_getUnit↩
Serial (
            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

buff	Text output buffer.
len	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, XM, XS, XA

Parameters

buff	data buffer.
length	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

bootComplete	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 CCAUXDLL_API void CCAUXDLL_CALLING_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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::About_setUser↩
EepromData (
            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, XM, XS, XA

Parameters

startpos	eeprom write start position.
buff	data buffer.
length	buffer length to write.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

selection	The type of voltage to get.
value	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 CCAUXDLL_API void CCAUXDLL_CALLING_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_3V3STBY);
output_voltage (pAdc, "VPMIC",       CrossControl::VOLTAGE_VPMIC);
output_voltage (pAdc, "VMAIN",       CrossControl::VOLTAGE_VMAIN);

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);
```

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

major	Major version number
minor	Minor version number
release	Release version number
build	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

major	Major version number
minor	Minor version number
release	Release version number
build	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

major	Major version number
minor	Minor version number
release	Release version number
build	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(column_width) << "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

major	Major version number
minor	Minor version number
release	Release version number
build	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_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

major	Major version number
minor	Minor version number
release	Release version number
build	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(column_width) << "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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

major	Major version number
minor	Minor version number
release	Release version number
build	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 CCAUXDLL_API void CCAUXDLL_CALLING_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);
```

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

averageWndSize	The average window size in nr of samples.
rejectWndSize	The reject window size in nr of samples.
rejectDeltaInLux	The reject delta in lux.
mode	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

bSoftTransitions	Soft transitions used?
k	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

status	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

status	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 CCAUXDLL_API 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

intensity	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↩
_getLedDimming (
            BACKLIGHTHANDLE ,
            CCStatus * status )
```

Get the current setting for Led dimming. If enabled, the function automatically dimms 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

status	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

status	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

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

Returns

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

Backlight_setAutomaticBLParams()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

bSoftTransitions	Use soft transitions?

Returns

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

Backlight_setIntensity()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

intensity	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Backlight↩
_setLedDimming (
            BACKLIGHTHANDLE ,
            CCStatus status )
```

Enable/disable Led dimming. If enabled, the function automatically dimms 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

status	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.

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

status	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_get↩
BatteryHWversion (
            BATTERYHANDLE ,
            char_t * buff,
            int32_t len )
```

Get battery hardware version (PCB revision).
Supported Platform(s): XM

Parameters

buff	Text output buffer.
len	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_get↩
BatterySerial (
            BATTERYHANDLE ,
            char_t * buff,
            int32_t len )
```

Get battery serial number.
Supported Platform(s): XM

Parameters

buff	Text output buffer.
len	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,buf, sizeof(buf));
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: " << buf << 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

major	Major version number
minor	Minor version number
release	Release version number
build	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_get↩
BatteryTemp (
            BATTERYHANDLE ,
            int16_t * temperature )
```

Get battery temperature.
Supported Platform(s): XM

Parameters

temperature	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

batteryVoltagePercent	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

errorCode	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_get↩
MinMaxTemp (
            BATTERYHANDLE ,
            int16_t * minTemp,
            int16_t * maxTemp )
```

Get temperature interval of the battery.
Supported Platform(s): XM

Parameters

minTemp	Minimum measured temperature.
maxTemp	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_get↩
PowerSource (
            BATTERYHANDLE ,
            PowerSource * status )
```

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

Parameters

status	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_get↩
Timer (
            BATTERYHANDLE ,
            BatteryTimerType * times )
```

Get battery diagnostic timer.
Supported Platform(s): XM

Parameters

times	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
    << "Total run time on battery power=" << times.TotRunTimeBattery*60 << " min(s)" << std::endl
    << "Total run time below -20C=" << times.RunTime_m20 << " min(s)" << std::endl
    << "Total run time -20-0C=" << times.RunTime_m20_0 << " min(s)" << std::endl
    << "Total run time 0-40C=" << times.RunTime_0_40 << " min(s)" << std::endl
    << "Total run time 40-60C=" << times.RunTime_40_60 << " min(s)" << std::endl
    << "Total run time 60-70C=" << times.RunTime_60_70 << " min(s)" << std::endl
    << "Total run time 70-80C=" << times.RunTime_70_80 << " min(s)" << std::endl
    << "Total run time above 80C=" << times.RunTime_Above80 << " min(s)" << std::endl;
}
```

Battery_isBatteryPresent()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Battery_is↩
BatteryPresent (
            BATTERYHANDLE ,
            bool * batteryIsPresent )
```

Is an external battery connected?
Supported Platform(s): XM

Parameters

batteryIsPresent	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 CCAUXDLL_API void CCAUXDLL_CALLING_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);
```

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

time	Time (ms) to buzz.
blocking	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

frequency	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

volume	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

trigger	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_getScaled↩ Volume 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

volume	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 CCAUXDLL_API 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 CCAUXDLL_API 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

frequency	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

volume	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

trigger	Status to set.

Returns

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

Buzzer_setVolume()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Buzzer_set↩
Volume (
            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

volume	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;
}
```

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

net	CAN net (1-4) to get settings for.
baudrate	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_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, XM

Parameters

net	CAN net (1-4) to get settings for.
frameType	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.
Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

net	CAN net (1-4).
baudrate	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_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, XM

Parameters

net	CAN net (1-4).
frameType	CAN frameType

Returns

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

CfgIn_getCfgInMode()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CfgIn_getCfg↩
InMode (
            CFGINHANDLE ,
            uint8_t channel,
            CfgInModeEnum * get_mode )
```

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

Parameters

channel	Which configurable input channel to use, 1-2 (VC), 1-8 (VA) or 1 (VS), corresponding to physical input channel

Parameters

get_mode	Storage container for retreived 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 CCAUXDLL_API 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 modes other than 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_FR↩ EQ_PD_10V - sample_value in Hz CFGIN_FREQ_PD_32V - sample_value in Hz C↩ FGIN_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_FR↩
EQ_PU_10V - sample_value in Hz CFGIN_FREQ_PU_32V - sample_value in Hz
VS: Port 1 only - sample_value in Hz
Supported Platform(s): VA, VS, VI2

Parameters

channel	Which configurable input channel to use, 1 through 8, corresponding to physical input channel
frequency	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_getMin↩
FrequencyThreshold (
            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 CFGIN_FREQ_PD_5V, CFGIN_FREQ_P↩
D_10V, CFGIN_FREQ_PD_32V, CFGIN_FREQ_F_5V, CFGIN_FREQ_F_10V, CFGI↩
N_FREQ_F_32V, CFGIN_FREQ_PU_5V, CFGIN_FREQ_PU_10V, CFGIN_FREQ_P↩
U_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

channel	Which configurable input channel to use, 1-2 (VC), 1-8 (VA) or 1 (VS), corresponding to physical input channel
frequency	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_getPwm↩
Value (
              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_PULL↩ DOWN).

Supported Platform(s): VC, VA, VI2

Parameters

channel	Which configurable input channel to use, 1 or 2, corresponding to physical input channel
frequency	Read signal frequency, 0.0 - 50000.0 Hz
duty_cycle	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 plaform: 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 µA, accuracy ± 0.5% ± 50 µA (typical) CFGIN_VOLTAGE_5V - sample_value in mV, accuracy ± 0.5% ± 5 mV (typical) CFGIN_VOLTAGE_10V - sample_value in mV, accuracy ± 0.5% ± 10 mV (typical) CFGIN_VOLTAGE_32V - sample_value in mV, accuracy ± 0.5% ± 32 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 ± 0.5% ± 5 Ohm (typical) CFGIN_RESISTANCE_500 - sample_value in 0.1 Ohm/bit, accuracy ± 0.5% ± 0.5 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, ports 1-4 only 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 (0.1Ohm steps)

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

Parameters

channel	Which configurable input channel to use, 1-2 (VC) or 1-8 (VA), corresponding to physical input channel
sample_value	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 CCAUXDLL_API 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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CfgIn_setCfg↩
InMode (
            CFGINHANDLE ,
            uint8_t channel,
            CfgInModeEnum set_mode )
```

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

Parameters

channel	Which configurable input channel to use, 1-2 (VC), 1-8 (VA) or 1 (VS), corresponding to physical input channel
set_mode	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CfgIn_setFrequency↩
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, VI2

Parameters

level	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::CfgIn_setMin↩
FrequencyThreshold (
             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 CFGIN_FREQ_PD_5V, CFGIN_FREQ_PD_10V, C↩ FGIN_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

channel	Which configurable input channel to use, 1-2 (VC), 1-8 (VA) or 1 (VS), corresponding to physical input channel
frequency	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:

```
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;
}
```

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

button_number	Which button to configure (1-MAX_BUTTONS)
button_config	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, XM, XS, XA

Parameters

status	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

output	Select power output (PowerOutput1 or PowerOutput2) .
enabled	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, XM

Parameters

status	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

triggertime	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

triggertime	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

temperature	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

duration	Long beep duration (ms)
frequency	Long beep frequency (Hz)
volume	Long beep volume (0-2000)

Returns

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

Config_getLongButtonPressAction()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

action	The configured action.

Returns

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

Config_getNextBootMode()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get↩
NextBootMode (
            CONFIGHANDLE ,
            BootModeEnum * mode )
```

Get BootMode for next boot
Supported Platform(s): VS, VI2, V700

Parameters

mode	Next BootMode

Returns

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

Config_getOnOffSigAction()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get↩
OnOffSigAction (
            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

action	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_get↩
OnOffSignalState (
            CONFIGHANDLE ,
            CCStatus * enabled )
```

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

Parameters

enabled	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_get↩
OnOffTriggerMode (
            CONFIGHANDLE ,
            ConfigOnOffTriggerMode * mode )
```

Get OnOff/Ignition/KeySwitch signal trigger mode.
Supported Platform(s): XM9, XL5, XA, XS, VC, VA, VS, VI2, V700

Parameters

mode	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_get↩
OSAliveMonitoring (
```

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

Get OS Alive Monitoring
Supported Platform(s): VS, VI2, V700

Parameters

enabled	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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 Startup↩ TriggerConfig 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

status	Enabled/Disabled

Returns

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

Config_getRS485Enabled()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get↩
RS485Enabled (
            CONFIGHANDLE ,
            RS4XXPort port,
            bool * enabled )
```

Get RS485 mode configuration for RS4XX port.
Supported Platform(s): XA, XS

Parameters

port	RS4XX port (RS4XXPort1-4)
enabled	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_get↩
ShortBeepSettings (
            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

duration	Short beep duration (ms)
frequency	Short beep frequency (Hz)
volume	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_get↩
ShortButtonPressAction (
            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, VI2

Parameters

action	The configured action.

Returns

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

Config_getStartupTriggerConfig()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

config	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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, XM

Parameters

voltage	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

maxTime	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_get↩
VideoStartupPowerConfig (
            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, XM, XS, XA

Parameters

config	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 CCAUXDLL_API 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:

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

conf_example(pConfig);

Config_release(pConfig);
```

Config_setButtonFunction()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set↩
ButtonFunction (
            CONFIGHANDLE ,
            uint8_t button_number,
            ButtonConfigEnum button_config )
```

Set button function configuration
Supported Platform(s): VC, VA, VS, VI2

Parameters

button_number	Which button to configure (1-MAX_BUTTONS)
button_config	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set↩
CanStartupPowerConfig (
            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, XM, XS, XA

Parameters

status	Enabled/Disabled

Returns

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

Config_setDigPowerOutputStartupConfig()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Config_set↩
DigPowerOutputStartupConfig (
            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

output	Select power output (PowerOutput1 or PowerOutput2).
enabled	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, XM

Parameters

status	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

triggertime	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

triggertime	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_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

temperature	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

duration	Long beep duration (ms)
frequency	Long beep frequency (Hz)
volume	Long beep volume (0-2000)

Returns

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

Config_setLongButtonPressAction()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

action	The action to set.

Returns

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

Config_setNextBootMode()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

mode	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_set↩
OnOffSigAction (
            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

action	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_set↩
```

```
OnOffTriggerMode (
            CONFIGHANDLE ,
            ConfigOnOffTriggerMode mode )
```

Set OnOff/Ignition/KeySwitch signal trigger mode.
Supported Platform(s): XM9, XL5, XA, XS, VC, VA, VS, VI2, V700

Parameters

mode	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_set↩
OSAliveMonitoring (
            CONFIGHANDLE ,
            CCStatus enabled )
```

Set OS Alive Monitoring
Supported Platform(s): VS, VI2, V700

Parameters

enabled	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_set↩
PowerOnStartup (
            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 Startup↩ TriggerConfig 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

status	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

port	RS4XX port (RS4XXPort1-4)
enabled	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

duration	Short beep duration (ms)
frequency	Short beep frequency (Hz)

Parameters

volume	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_set↩
ShortButtonPressAction (
            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

action	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_set↩
StartupTriggerConfig (
            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. See enum TriggerConf for more details.

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

Parameters

conf	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_set↩
StartupVoltageConfig (
            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, XM

Parameters

voltage	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_set↩
SuspendMaxTime (
            CONFIGHANDLE ,
            uint16_t maxTime )
```

Set suspend mode maximum time.

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

Parameters

maxTime	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_set↩
VideoStartupPowerConfig (
            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

config	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.

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

errorCode	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 CCAUXDLL_API 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

minTemp	Minimum measured PCB temperature.
maxTemp	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 CCAUXDLL_API 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

temperature	PCB Temperature in degrees Celsius.

Returns

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

Diagnostic_getPMTemp()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↩
_getPMTemp (
            DIAGNOSTICHANDLE ,
            uint8_t index,
            int16_t * temperature,
            JidaSensorType * jst )
```

Get Processor Module temperature. This temperature is read from the Kontron J↩ IDA 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

index	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. CCpliot XM 2.0 supports only one sensor, CPU temperature.

Supported Platform(s): XL, XM

Parameters

temperature	Temperature in degrees Celsius.
jst	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

powerCycles	Total number of power cycles.

Returns

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

Diagnostic_getShutDownReason()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Diagnostic↩
_getShutDownReason (
            DIAGNOSTICHANDLE ,
            uint16_t * reason )
```

Get shutdown reason.
Supported Platform(s): XL, XM

Parameters

reason	See DiagnosticCodes.h for shutdown codes.

Returns

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

Diagnostic_getSSTemp()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

temperature	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

reason	See DiagnosticCodes.h 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

times	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:

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

diagnostic_example(pDiagnostic);

Diagnostic_release(pDiagnostic);
```

DigIO_getDigIO()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::DigIO_getDig↩
IO (
            DIGIOHANDLE ,
            uint8_t * status )
```

Get Digital inputs.
Supported Platform(s): XL, XM, XS, XA

Parameters

status	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_getDig↩
PowerOutput (
             DIGIOHANDLE ,
             PowerOutput output,
             CCStatus * enabled,
             uint8_t * status )
```

Get digital power output status.
Supported Platform(s): VS

Parameters

output	Which output to read
enabled	State of the power output signal.
status	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 0, this indicates an error (1 is no error).

If the signal state is Enabled (high), the following error conditions gererates 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 CCAUXDLL_API void CCAUXDLL_CALLING_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_setDig↩
IO (
            DIGIOHANDLE ,
            uint8_t state )
```

Set Digital outputs.
Supported Platform(s): XA, XS

Parameters

state	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_setDig↩
PowerOutput (
            DIGIOHANDLE ,
            PowerOutput output,
            CCStatus enabled )
```

Set Digital power outputs.
Supported Platform(s): VS

Parameters

output	The output to set
enabled	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:

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

status	The current status of the upgrade operation.
blocking	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_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

filename	Path and filename to the .mcs file to program.

Parameters

blocking	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 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

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code. 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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

filename	Path and filename to the .hex file to program.
blocking	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.

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 CCAUXDLL_API 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

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code. 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

filename	Path and filename to the .hex file to program.
blocking	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.

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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

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

Returns

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

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

red	Red color intensity 0-0x0F.
green	Green color intensity 0-0x0F.
blue	Blue color intensity 0-0x0F.

Parameters

frequency	LED blink frequency (0.2-50 Hz).
dutyCycle	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_get↩
Color (
            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

red	Red color intensity 0-0x0F.
green	Green color intensity 0-0x0F.
blue	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_get↩
EnabledDuringStartup (
            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

status	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

idleTime	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

nrOfPulses	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

offTime	Time in 10ms increments.

Returns

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

FrontLED_getOnTime()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

onTime	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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 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

red	Red color intensity 0-0x0F.
green	Green color intensity 0-0x0F.
blue	Blue color intensity 0-0x0F.
frequency	LED blink frequency (0.2-50 Hz).
dutyCycle	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 CCAUXDLL_API 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

frequency	LED blink frequency (0.2-50 Hz).
dutyCycle	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

color	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:

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

led_example(pFrontLED);

FrontLED_release(pFrontLED);
```

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

red	Red color intensity 0-0x0F.
green	Green color intensity 0-0x0F.
blue	Blue color intensity 0-0x0F.
frequency	LED blink frequency (0.2-50 Hz).
dutyCycle	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_set↩
Color (
            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

red	Red color intensity 0-0x0F.
green	Green color intensity 0-0x0F.
blue	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_set↩
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

status	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_set↩
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

idleTime	Time in 100ms.

Returns

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

FrontLED_setNrOfPulses()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::FrontLED_set↩
NrOfPulses (
            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

nrOfPulses	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_set↩
Off (
            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_set↩
OffTime (
            FRONTLEDHANDLE ,
            uint8_t offTime )
```

Set front LED off time.
Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

offTime	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_set↩
OnTime (
            FRONTLEDHANDLE ,
            uint8_t onTime )
```

Set front LED on time.
Supported Platform(s): XL, XL5, XM, XM9, XS, XA, VC, VA, VS, VI2, V700

Parameters

onTime	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_set↩
PostBootLEDConfig (
            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 L↩ ED 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

red	Red color intensity 0-0x0F.
green	Green color intensity 0-0x0F.
blue	Blue color intensity 0-0x0F.
frequency	LED blink frequency (0.2-50 Hz).
dutyCycle	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_set↩
Signal (
            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

frequency	LED blink frequency (0.2-50 Hz).
dutyCycle	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_set↩
StandardColor (
            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

color	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;
}
```

GetAbout()

```
EXTERN_C CCAUXDLL_API ABOUTHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get↩
About (
            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);
```

GetAdc()

```
EXTERN_C CCAUXDLL_API ADCHANDLE CCAUXDLL_CALLING_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_3V3STBY);
output_voltage (pAdc, "VPMIC",      CrossControl::VOLTAGE_VPMIC);
output_voltage (pAdc, "VMAIN",      CrossControl::VOLTAGE_VMAIN);

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);
```

GetAuxVersion()

```
EXTERN_C CCAUXDLL_API AUXVERSIONHANDLE CCAUXDLL_CALLING_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);
```

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);
```

GetBattery()

```
EXTERN_C CCAUXDLL_API BATTERYHANDLE CCAUXDLL_CALLING_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);
```

GetBuzzer()

```
EXTERN_C CCAUXDLL_API BUZZERHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get↩
Buzzer (
            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_beeps(pBuzzer);

Buzzer_release(pBuzzer);
```

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);
```

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);
```

GetConfig()

```
EXTERN_C CCAUXDLL_API CONFIGHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get↩
Config ( )
```

Video channel 4 config

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);
```

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);
```

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);
```

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, XM, XS, XA, VS, VI2

Parameters

errCode	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, XM, XS, XA, VS, VI2

Parameters

errCode	An error code for which

Returns

String description of an error code.

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);
```

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);
```

GetHwErrorStatusStringA()

```
EXTERN_C CCAUXDLL_API char_t const* CCAUXDLL_CALLING_CONV CrossControl::Get↩
HwErrorStatusStringA (
            uint16_t errCode )
```

Get a string description of an error code returned from getHwErrorStatus.
Supported Platform(s): All

Parameters

errCode	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

errCode	An error code for which to get a string description.

Returns

String description of an error code.

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);
```

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);
```

GetPowerMgr()

```
EXTERN_C CCAUXDLL_API POWERMGRHANDLE CCAUXDLL_CALLING_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);
```

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);
```

GetSmart()

```
EXTERN_C CCAUXDLL_API SMARTHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get↩
Smart (
            void  )
```

Factory function that creates instances of the Smart object.
Supported Platform(s): XL, XM

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);
```

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);
```

GetStartupReasonStringA()

```
EXTERN_C CCAUXDLL_API char_t const* CCAUXDLL_CALLING_CONV CrossControl::Get↩
StartupReasonStringA (
            uint16_t code )
```

Get a string description of a startup reason code returned from getStartupReason.
Supported Platform(s): All

Parameters

code	A code for which to get a string description.

Returns

String description of a code.

GetStartupReasonStringW()

```
EXTERN_C CCAUXDLL_API 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

code	A code for which to get a string description.

Returns

String description of a code.

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);
```

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);
```

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.

GetVideo()

```
EXTERN_C CCAUXDLL_API VIDEOHANDLE CCAUXDLL_CALLING_CONV CrossControl::Get↩
Video (
            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.

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

value	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

value	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 ch1 are raw ADC values read from a TAOS TSL2550 lightsensor.

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

Parameters

value	Illuminance value (Lux).
ch0	Channel0 value. (Not applicable on VC platform - always 0)
ch1	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 CCAUXDLL_API 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

range	Operating range. RangeStandard or RangeExtended.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Lightsensor_release()

```
EXTERN_C CCAUXDLL_API 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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

range	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

averageWndSize	The average window size in nr of samples.
rejectWndSize	The reject window size in nr of samples.
rejectDeltaInLux	The reject delta in lux.
mode	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 backlihgt 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;
}
```

Power_ackPowerRequest()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_ackPower↩
Request (
            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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_getB↩
LPowerStatus (
            POWERHANDLE ,
            CCStatus * status )
```

Get backlight power status.
Supported Platform(s): XL, XM

Parameters

status	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_getButton↩
PowerTransitionStatus (
            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

status	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_getCanOCDStatus()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_getCan↩
OCDStatus (
            POWERHANDLE ,
            OCDStatus * 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, XM, XS, XA

Parameters

status	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;
OCDStatus ocdstatus;
err = Power_getCanOCDStatus(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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_getCan↩
PowerStatus (
            POWERHANDLE ,
            CCStatus * status )
```

Get can power status.
Supported Platform(s): XL, XM, XS, XA

Parameters

status	Can power status.

Returns

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

Power_getExtFanPowerStatus()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Power_getExt↩
FanPowerStatus (
            POWERHANDLE ,
            CCStatus * status )
```

Get external fan power status.
Supported Platform(s): XL, XM

Parameters

status	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_getVideo↩
OCDStatus (
            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

status	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_getVideo↩
PowerStatus (
            POWERHANDLE ,
            uint8_t * videoStatus )
```

Get Video power status.
Supported Platform(s): XL, XM, XS, XA

Parameters

videoStatus	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_setB↩
LPowerStatus (
              POWERHANDLE ,
              CCStatus status )
```

Set backlight power status.
Supported Platform(s): XL, XM

Parameters

status	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_setCan↩
PowerStatus (
```

```
            POWERHANDLE ,
            CCStatus status )
```

Set can power status.
Supported Platform(s): XL, XM, XS, XA

Parameters

status	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_setExt↩
FanPowerStatus (
            POWERHANDLE ,
            CCStatus status )
```

Set external fan power status.
Supported Platform(s): XL, XM

Parameters

status	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

status	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.

PowerMgr_getConfiguration()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PowerMgr_get↩
Configuration (
            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

conf	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 CCAUXDLL_API 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

status	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 appp
      }
    case SuspendPending:
      {
        os_shutdown = false;

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

        // Make sure to clear cin buffer before read
        std::cin.clear();
        std::cin.ignore(100,'\n');
        cin.get();
        cout << "Signalling that app is ready..."  << endl;
        err = 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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::PowerMgr_has↩
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 appp
      }
    case SuspendPending:
      {
        os_shutdown = false;

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

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

conf	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 CCAUXDLL_API void CCAUXDLL_CALLING_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 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_setAppReadyForSuspendOrShutdown()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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 appp
      }
    case SuspendPending:
      {
        os_shutdown = false;

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

        // Make sure to clear cin buffer before read
        std::cin.clear();
        std::cin.ignore(100,'\n');
        cin.get();
        cout << "Signalling that app is ready..."  << endl;
        err = 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;
  }
}
```

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

channel	Which channel to get value from There are two output channels, 1 or 2.
duty_cycle	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

channel	Which channel to set There are two output channels, 1 or 2.
frequency	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

status	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 occurrs. 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 CCAUXDLL_API void CCAUXDLL_CALLING_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_set↩
PWMOutOff (
            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

channel	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_set↩
PWMOutputChannelDutyCycle (
            PWMOUTHANDLE ,
            uint8_t channel,
            uint8_t duty_cycle )
```

Set PWM Output Duty cycle for a channel

Supported Platform(s): VC, VA, VI2

Parameters

channel	Which channel to set There are two output channels, 1 or 2.
duty_cycle	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

channel	Which channel to set There are two output channels, 1 or 2.
frequency	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;
}
```

Smart_getDeviceSerial()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Smart_getDevice↩
Serial (
              SMARTHANDLE ,
              char_t * buff,
              int32_t len )
```

Get serial number of the secondary storage device.
Supported Platform(s): XL, XM

Parameters

buff	Text output buffer.
len	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Smart_getDevice↩
Serial2 (
              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

buff	Text output buffer.
len	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_E↩ XIST 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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Smart_getInitial↩
Time (
              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

Parameters

time	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_getInitial↩
Time2 (
              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

time	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_E↩ XIST 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_getRemaining↩
LifeTime (
              SMARTHANDLE ,
              uint8_t * lifetimepercent )
```

Get remaining lifetime of the secondary storage device.
Supported Platform(s): XL, XM

Parameters

lifetimepercent	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

lifetimepercent	The expected remaining lifetime (0..100%).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code. ERR_CODE_NOT_E↩ XIST 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

Returns

-

Example Usage:

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

show_card_data(pSmart);

Smart_release(pSmart);
```

SoftKey_getBacklightEnabledDuringStartup()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

status	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_get↩
BacklightIdleTime (
            SOFTKEYHANDLE ,
            uint8_t * idleTime )
```

Get SoftKey Backlight IDLE time.

Supported Platform(s): VI2

Parameters

idleTime	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

key	Key that's intensity is read. Start index 1.
intensity	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

nrOfPulses	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

offTime	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

onTime	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

frequency	Backlight blink frequency (0.2-50 Hz).
dutyCycle	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

bootIntensity	Backlight intensity (0-100%) during system boot.
bootFrequency	Backlight blink frequency (0.2-50 Hz) during system boot.
bootDutyCycle	Backlight on duty cycle (0-100%) during system boot.
postBootIntensity	Backlight intensity (0-100%) after system boot.
postBootFrequency	Backlight blink frequency (0.2-50 Hz) after system boot.
postBootDutyCycle	Backlight on duty cycle (0-100%) after system boot.
postBootConfig	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

keys	Array of Keys that's intensity to set. Key indexing starts at number 1.
intensities	Array of intensities for the keys listed above. Inetnsity value in range 0 - 100%
array_size	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:

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

value	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 CCAUXDLL_API void CCAUXDLL_CALLING_CONV CrossControl::SoftKey_release
(
            SOFTKEYHANDLE  )
```

Delete the SOFTKEY object.
Supported Platform(s): VI2

Returns

-

Example Usage:

SoftKey_setBacklightEnabledDuringStartup()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

status	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

idleTime	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

key	Key that's intensity is set. Start index 1.
intensity	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

nrOfPulses	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

offTime	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↩
BacklightOnTime (
            SOFTKEYHANDLE ,
            uint8_t onTime )
```

Set SoftKey Backlight ON time.
Supported Platform(s): VI2

Parameters

onTime	Time in 10ms increments. 0 = off

Returns

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

Example Usage:

SoftKey_setBacklightSignal()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↩
BacklightSignal (
            SOFTKEYHANDLE ,
            float64_t frequency,
            uint8_t dutyCycle )
```

Set SoftKey Backlight signal.
Supported Platform(s): VI2

Parameters

frequency	SoftKey Backlight blink frequency (0.2-50 Hz).
dutyCycle	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

bootIntensity	Backlight intensity (0-100%) during system boot.
bootFrequency	Backlight blink frequency (0.2-50 Hz) during system boot.
bootDutyCycle	Backlight on duty cycle (0-100%) during system boot.
postBootIntensity	Backlight intensity (0-100%) after system boot.
postBootFrequency	Backlight blink frequency (0.2-50 Hz) after system boot.
postBootDutyCycle	Backlight on duty cycle (0-100%) after system boot.
postBootConfig	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::SoftKey_set↩
MultipleBacklightIntensities (
            SOFTKEYHANDLE ,
            uint8_t * keys,
            uint8_t * intensities,
            uint8_t array_size )
```

Set Multiple SoftKey backlight intensities.

Supported Platform(s): VI2

Parameters

keys	Array of Keys that's intensity to set. Key indexing starts at number 1.
intensities	Array of intensities for the keys listed above. Intensity value in range 0 - 100%
array_size	Number of key/intensity - pairs

To set all keys to same intensitity. 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;
}
```

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

status	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

status	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

status	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_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

status	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

status	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

status	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

status	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↩
_getTelematicsAvailable (
            TELEMATICSHANDLE ,
            CCStatus * status )
```

Is a telematics add-on card installed?
Supported Platform(s): XM, XA, XS

Parameters

status	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Telematics↩
_getWLANPowerStatus (
            TELEMATICSHANDLE ,
            CCStatus * status )
```

Get WLAN power status.
Supported Platform(s): XM, XA, XS

Parameters

status	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

status	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

status	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

status	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

status	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

status	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

status	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

status	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

status	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

status	WLAN power status.

Returns

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

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

param	The setting to get.
data	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

config	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

time	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

param	The setting to set.
data	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

config	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

time	The right click time, in milliseconds.

Returns

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

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

finished	Is current point finished?
pointNr	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

param	Config parameter
value	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

mode	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

pointNr	Calibration point number (1 to total number of points)

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

TouchScreenCalib_setConfigParam()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↩
```

```
Calib_setConfigParam (
            TOUCHSCREENCALIBHANDLE ,
            CalibrationConfigParam param,
            uint16_t value )
```

Set calibration config parameters
Supported Platform(s): XL, XM, XS, XA

Parameters

param	Config parameter
value	parameter value

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

TouchScreenCalib_setMode()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::TouchScreen↩
Calib_setMode (
            TOUCHSCREENCALIBHANDLE ,
            CalibrationModeSettings mode )
```

Set mode of front controller.
Supported Platform(s): XL, XM, XS, XA

Parameters

mode	Selected calibration mode

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Video_activateSnapshot()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_activate↩
Snapshot (
            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

activate	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_create↩
Bitmap (
            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

bmpBuffer	Bitmap ram buffer allocated by the API, has to be deallocated with freeBmpBuffer() by the application.
bmpBufSize	Size of the returned bitmap buffer.
rawImgBuffer	Raw image buffer from takeSnapShotRaw.
rawImgBufSize	Size of the raw image buffer.
bInterlaced	Interlaced, if true the bitmap only contains every second line in the image, to save bandwidth.
bNTSCFormat	True if the video format in rawImageBuffer is NTSC format.

Returns

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

Video_freeBmpBuffer()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_free↩
BmpBuffer (
```

```
            VIDEOHANDLE ,
            char_t * bmpBuffer )
```

Free the memory allocated for BMP buffer.
Supported Platform(s): XL, XM (Windows)

Parameters

bmpBuffer	The bmp buffer to free.

Returns

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

Video_getActiveChannel()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getActive↩
Channel (
            VIDEOHANDLE ,
            VideoChannel * channel )
```

Get the current video channel.
Supported Platform(s): XL, XM, XS, XA, VC, VA

Parameters

channel	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getColor↩
Keys (
            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

rKey	Red value.
gKey	Green value.
bKey	Blue value.

Returns

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

Video_getCropping()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

top	Crop top (lines).
left	Crop left (lines).
bottom	Crop bottom (lines).
right	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_getDecoder↩
Reg (
            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

decoderRegister	Decoder Register Address.
registerValue	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_getDe↩
InterlaceMode (
            VIDEOHANDLE ,
            DeInterlaceMode * mode )
```

Get the deinterlace mode used when decoding the interlaced video stream.
Supported Platform(s): XL, XM

Parameters

mode	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getGraphics↩
Overlay (
            VIDEOHANDLE ,
            CCStatus * mode )
```

Get the current graphics overlaying mode.
Supported Platform(s): XA, XS, VC, VA

Parameters

mode	Overlay enable mode

Returns

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

Video_getMirroring()

```
EXTERN_C CCAUXDLL_API 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

mode	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getRaw↩
Image (
            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

width	Width of raw image.
height	Height of raw moving image, snapshot are 4 bytes less.
frameRate	Received video frame rate.

Returns

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

Video_getRotation()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_getRotation
(
            VIDEOHANDLE ,
            VideoRotation * rotation )
```

Get the current rotation of the video image.
Supported Platform(s): XA, XS, VC, VA, VS

Parameters

rotation	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

x	Horizontal scaling (0.25-4).
y	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

status	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_getVideo↩
Area (
            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

topLeftX	Top left X coordinate on screen.
topLeftY	Top left Y coordinate on screen.
bottomRigthX	Bottom right X coordinate on screen.
bottomRigthY	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_getVideo↩
Standard (
            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

standard	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

deviceNr	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

channel	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

rKey	Red key value.
gKey	Green key value.
bKey	Blue key value.

Returns

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

Video_setCropping()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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 horisontal 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 allways 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

top	Crop top (0-255 lines).
left	Crop left (0-127 lines).
bottom	Crop bottom (0-255 lines).
right	Crop right (0-127 lines).

Returns

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

Video_setDecoderReg()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

decoderRegister	Decoder Register Address.
registerValue	register value.

Returns

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

Video_setDeInterlaceMode()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_setDe↩
InterlaceMode (
            VIDEOHANDLE ,
            DeInterlaceMode mode )
```

Set the deinterlace mode used when decoding the interlaced video stream.
Supported Platform(s): XL, XM

Parameters

mode	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_setGraphics↩
Overlay (
            VIDEOHANDLE ,
            CCStatus mode )
```

Enable or disable overlaying of graphics on top of video.
Supported Platform(s): XA, XS, VC, VA

Parameters

mode	Overlay enable mode

Returns

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

Video_setMirroring()

```
EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_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

mode	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 CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV CrossControl::Video_setRotation
(
            VIDEOHANDLE ,
            VideoRotation rotation )
```

Set the current rotation of the video image.
Supported Platform(s): XA, XS, VC, VA, VS

Parameters

rotation	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 CCAUXDLL_API 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

x	Horizontal scaling (0.25-4).
y	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 CCAUXDLL_API 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

topLeftX	Top left X coordinate on screen.
topLeftY	Top left Y coordinate on screen.
bottomRightX	Bottom right X coordinate on screen.
bottomRightY	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

show	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

path	The file path to where the image should be stored.
bInterlaced	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 CCAUXDLL_API 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 takeSnap↩ ShotRaw and createBitMap. To be able to take a snapshot, the snapshot function has to be active.

Supported Platform(s): XL, XM (Windows)

Parameters

bmpBuffer	Bitmap ram buffer allocated by the API, has to be deallocated with freeBmpBuffer() by the application.
bmpBufSize	Size of the returned bitmap buffer.
bInterlaced	If true the bitmap only contains every second line in the image, to save bandwidth.
bNTSCFormat	True if the video format in rawImageBuffer 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 + line count $*$ row count $*$ 4. The size of the raw image is when interlaced = true 0x100 + line count $*$ row count $*$ 2.

To be able to take a snapshot, the snapshot function has to be active. This function is blocking until a new frame is available from the decoder. An error will be returned if the decoder doesn't return any frames before a timeout.

Supported Platform(s): XL, XM (Windows)

Parameters

rawImgBuffer	Buffer for image to be stored in.
rawImgBuffSize	Size of the buffer.
bInterlaced	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.

5.1.4 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 6

Data Structure Documentation

6.1 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

6.1.1 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

6.2 BuzzerSetup Struct Reference

`#include <CCAuxTypes.h>`

Data Fields

- uint16_t frequency
- uint16_t volume

6.2.1 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

6.3 FpgaLedTimingType Struct Reference

#include <CCAuxTypes.h>

Data Fields

- uint8_t ledNbr
- uint8_t onTime
- uint8_t offTime
- uint8_t idleTime
- uint8_t nrOfPulses

6.3.1 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

6.4 LedColorMixType Struct Reference

`#include <CCAuxTypes.h>`

Data Fields

- uint8_t red
- uint8_t green
- uint8_t blue

6.4.1 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

6.5 LedTimingType Struct Reference

`#include <CCAuxTypes.h>`

Data Fields

- uint8_t onTime
- uint8_t offTime
- uint8_t idleTime
- uint8_t nrOfPulses

6.5.1 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

6.6 received_video Struct Reference

`#include <CCAuxTypes.h>`

Data Fields

- uint16_t received_width
- uint16_t received_height
- uint8_t received_framerate

6.6.1 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

6.7 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

6.7.1 Detailed Description

Diagnostic timer data

6.7.2 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

6.8 UpgradeStatus Struct Reference

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

Data Fields

- enum UpgradeAction currentAction
- uint8_t percent
- eErr errorCode

6.8.1 Detailed Description

Upgrade Status

6.8.2 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

6.9 version_info Struct Reference

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

Data Fields

- uint8_t major
- uint8_t minor
- uint8_t release
- uint8_t build

6.9.1 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

6.10 video_dec_command Struct Reference

`#include <CCAuxTypes.h>`

Data Fields

- uint8_t decoder_register
- uint8_t register_value

6.10.1 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 7

File Documentation

7.1 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_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)

7.2 IncludeFiles/Adc.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * ADCHANDLE

Functions

- EXTERN_C CCAUXDLL_API ADCHANDLE CCAUXDLL_CALLING_CO↩ NV 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)

7.3 IncludeFiles/AuxVersion.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * AUXVERSIONHANDLE

Functions

- EXTERN_C CCAUXDLL_API AUXVERSIONHANDLE CCAUXDLL_CA↩ LLING_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)

7.4 IncludeFiles/Backlight.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * BACKLIGHTHANDLE

Functions

- EXTERN_C CCAUXDLL_API BACKLIGHTHANDLE CCAUXDLL_CAL↩ LING_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)

7.5 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_CALLIN↩ G_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)

7.6 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)

7.7 IncludeFiles/CanSetting.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * CANSETTINGHANDLE

Functions

- EXTERN_C CCAUXDLL_API CANSETTINGHANDLE CCAUXDLL_CAL↩ LING_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)

7.8 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_CO↩ NV GetErrorStringW (eErr errCode)

7.9 IncludeFiles/CCAuxTypes.h File Reference

Data Structures

- struct received_video
- struct video_dec_command
- struct version_info
- struct BuzzerSetup
- struct LedTimingType
- struct 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_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_IOCTRL_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_FAILE
  = 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 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_TR = 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_TRIGG }
- enum PowerOutput { PowerOutput1 = 0, PowerOutput2, PowerOutput3, PowerOutput4, PowerOutput5, PowerOutput6, PowerOutputMax }
- enum SystemMode { SYSTEMMODE_Startup = 0, SYSTEMMODE_StartupRescue = 1, SYSTEMMODE_StartupRescueFactoryRese = 2, SYSTEMMODE_NormalRunning = 3, SYSTEMMODE_RescueRunning = 4, SYSTEMMODE_RescueRunningFactoryReset = 5, SYSTEMMODE_Unknown = 6 }

7.9.1 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
```

7.10 IncludeFiles/CCPlatform.h File Reference

7.11 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)

7.12 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)

7.13 IncludeFiles/Diagnostic.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * DIAGNOSTICHANDLE

Functions

- EXTERN_C CCAUXDLL_API DIAGNOSTICHANDLE CCAUXDLL_CAL↩ LING_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)

7.14 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)

7.15 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)

7.16 IncludeFiles/FirmwareUpgrade.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * FIRMWAREUPGHANDLE

Functions

- EXTERN_C CCAUXDLL_API FIRMWAREUPGHANDLE CCAUXDLL_C↩ ALLING_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_getUpgradeStatus (FIRMWAREUPGHANDLE, UpgradeStatus *status, bool blocking)
- EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV FirmwareUpgrade_shutDown (FIRMWAREUPGHANDLE)

7.17 IncludeFiles/FrontLED.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * FRONTLEDHANDLE

Functions

- EXTERN_C CCAUXDLL_API FRONTLEDHANDLE CCAUXDLL_CALLI↩ NG_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)

7.18 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)

7.19 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)

7.20 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_CAL↩ LING_CONV GetPowerMgr (void)
- EXTERN_C CCAUXDLL_API void CCAUXDLL_CALLING_CONV PowerMgr_release (POWERMGRHANDLE)
- EXTERN_C CCAUXDLL_API eErr CCAUXDLL_CALLING_CONV PowerMgr_registerControlledSuspendOr (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)

7.21 IncludeFiles/PWMOut.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * PWMOUTHANDLE

Functions

- EXTERN_C CCAUXDLL_API PWMOUTHANDLE CCAUXDLL_CALLIN↩ G_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)

7.22 IncludeFiles/Releasenotes.dox File Reference

7.23 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)

7.24 IncludeFiles/SoftKey.h File Reference

Namespaces

- CrossControl

Typedefs

- typedef void * SOFTKEYHANDLE

Functions

- EXTERN_C CCAUXDLL_API SOFTKEYHANDLE CCAUXDLL_CALLIN↩ G_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_getBacklightEnabledDuringStar (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)

7.25 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_getWLANStartUpPowerStatu (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_setWLANStartUpPowerStatu (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)

7.26 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_C↩ ALLING_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)

7.27 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 CCAUX↩
  DLL_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)

7.28 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 ∗bottom↩ RigthX, 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 b↩ Interlaced)
- 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 *raw↩ ImgBuffer, 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)