CCAux

1.6.4.0

Tue Apr 16 2013

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.

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 About
- struct Adc
- struct AuxVersion
- struct Backlight
- struct BatteryTimerType
- struct Battery
- struct Buzzer
- struct CanSetting
- struct received_video
- struct video_dec_command
- struct version_info
- struct BuzzerSetup
- struct LedTimingType
- struct LedColorMixType
- struct TimerType
- struct UpgradeStatus
- struct Config
- struct Diagnostic
- struct DigIO
- struct FirmwareUpgrade
- struct FrontLED
- struct Lightsensor
- struct Power
- struct PowerMgr
- struct Smart
- struct Telematics
- struct TouchScreen
- struct TouchScreenCalib
- struct Video

Typedefs

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

Enumerations

- enum VoltageEnum { VOLTAGE_24VIN = 0, VOLTAGE_24V, VOLTAGE_12V, VOLTAGE_12-VID, VOLTAGE_5V, VOLTAGE_3V3, VOLTAGE_VTFT, VOLTAGE_5VSTB, VOLTAGE_1V9, VOLTAGE_1V8, VOLTAGE_1V5, VOLTAGE_1V2, VOLTAGE_1V05, VOLTAGE_1V0, VOLTAGE_0V9, VOLTAGE_VREF_I-NT }
- enum ChargingStatus { ChargingStatus_NoCharge = 0, ChargingStatus_Charging = 1, ChargingStatus-_FullyCharged = 2, ChargingStatus_TempLow = 3, ChargingStatus_TempHigh = 4, ChargingStatus_Unknown = 5 }
- enum PowerSource { PowerSource_Battery = 0, PowerSource_ExternalPower = 1 }
- enum ErrorStatus { ErrorStatus_NoError = 0, ErrorStatus_ThermistorTempSensor = 1, ErrorStatus-_SecondaryTempSensor = 2, ErrorStatus_ChargeFail = 3, ErrorStatus_Overcurrent = 4, ErrorStatus_Init = 5 }
- enum LightSensorOperationRange { RangeStandard = 0, RangeExtended = 1 }
- enum LightSensorSamplingMode { SamplingModeStandard = 0, SamplingMode-Extended, SamplingModeAuto }
- enum CCStatus { Disabled = 0, Enabled = 1 }
- enum eErr { ERR_SUCCESS = 0, ERR_OPEN_FAILED = 1, ERR_NOT_SUPPORTED = 2, ERR_UNKNOWN_FEATURE = 3, ERR_DATATYPE_MISMATCH = 4, ERR_CODE_NOT_EXIST = 5, ERR_-BUFFER_SIZE = 6, ERR_IOCTRL_FAILED = 7, ERR_INVALID_DATA = 8, ERR_INVALID_PARAMETER = 9, ERR_CRE-ATE_THREAD = 10, ERR_IN_PROGRESS = 11, ERR_CHECKSUM = 12, ERR_INIT_FAILED = 13, ERR_VERIFY_FAILED = 14, ERR_DEVICE_READ_DATA_FAILED = 15, ERR_DEVICE_WRITE_DATA_FAILED = 16, ERR_COMMAND_FAILED = 17, ERR_EEPROM = 18, ERR_JIDA_TEMP = 19, ERR_AVERAGE_CALC_STARTED = 20, ERR_NOT_RUNNING = 21, ER-R_I2C_EXPANDER_READ_FAILED = 22, ERR_I2C_EXPANDER_WRITE-_FAILED = 23, ERR_I2C_EXPANDER_INIT_FAILED = 24, ERR_NEWER_SS_VERSION-_REQUIRED = 25, ERR_NEWER_FPGA_VERSION_REQUIRED = 26, ER-R_NEWER_FRONT_VERSION_REQUIRED = 27, ERR_TELEMATICS_GPRS_NOT_AVAILABLE = 28, ERR_TELEMATICS-

_WLAN_NOT_AVAILABLE = 29, ERR_TELEMATICS_BT_NOT_AVAIL-ABLE = 30, ERR_TELEMATICS_GPS_NOT_AVAILABLE = 31, ERR_MEM_ALLOC_FAIL = 32 }

- enum DeInterlaceMode { DeInterlace_Even = 0, DeInterlace_Odd = 1, DeInterlace-_BOB = 2 }
- enum VideoChannel { Analog_Channel_1 = 0, Analog_Channel_2 = 1, Analog-_Channel_3 = 2, Analog_Channel_4 = 3 }
- enum videoStandard {
  STD_M_J_NTSC = 0, STD_B_D_G_H_I_N_PAL = 1, STD_M_PAL = 2, ST-D_PAL = 3,
  STD_NTSC = 4, STD_SECAM = 5 }
- enum CanFrameType { FrameStandard, FrameExtended, FrameStandardExtended }
- enum TriggerConf { Front_Button_Enabled = 1, OnOff_Signal_Enabled = 2, Both_Button_And_Signal_Enabled = 3 }
- enum PowerAction { NoAction = 0, ActionSuspend = 1, ActionShutDown = 2 }
- enum ButtonPowerTransitionStatus {
  BPTS_No_Change = 0, BPTS_ShutDown = 1, BPTS_Suspend = 2, BPTS_-Restart = 3,
  BPTS_BtnPressed = 4, BPTS_BtnPressedLong = 5, BPTS_SignalOff = 6 }
- enum JidaSensorType {
  TEMP_CPU = 0, TEMP_BOX = 1, TEMP_ENV = 2, TEMP_BOARD = 3,
  TEMP_BACKPLANE = 4, TEMP_CHIPSETS = 5, TEMP_VIDEO = 6, TEM-P_OTHER = 7 }
- enum UpgradeAction {
  UPGRADE_INIT, UPGRADE_PREP_COM, UPGRADE_READING_FILE, U-PGRADE_CONVERTING_FILE,
  UPGRADE_FLASHING, UPGRADE_VERIFYING, UPGRADE_COMPLET-E, UPGRADE_COMPLETE_WITH_ERRORS }
- enum CCAuxColor {
  RED = 0, GREEN, BLUE, CYAN,
  MAGENTA, YELLOW, UNDEFINED_COLOR }
- enum startupReasonCodes {
  startupReasonCodeUndefined = 0x0000, startupReasonCodeButtonPress = 0x0055,
  startupReasonCodeExtCtrl = 0x00AA, startupReasonCodeMPRestart = 0x00F0,
  startupReasonCodePowerOnStartup = 0x000F }
- enum shutDownReasonCodes { shutdownReasonCodeNoError = 0x001F }
- enum hwErrorStatusCodes {
  errCodeNoErr = 0, errCodeFPGACONFReadErr = 1, errCodeFPGACONFUnexp-Val = 2, errCodeCBRESETReadErr = 3,
  errCodeSUS3ReadErr = 4, errCodeSUS4ReadErr = 5, errCodeSUS5ReadErr = 6, errCodePG5VSTBYReadErr = 7,
  errCodePG5VSTBYUnexpVal = 8, errCodeCANPWROKReadErr = 9, errCode-VIDPWROKReadErr = 10, errCodeLVDSBLENReadErr = 11,
  errCodeLVDSVDDENReadErr = 12, errCodeEXTCTRLONReadErr = 13, err-CodeFPBTNONReadErr = 14, errCode24VReadErr = 15,
  errCode24VOutOfLimits = 16, errCode24VINReadErr = 17, errCode24VIN-OutOfLimits = 18, errCode12VReadErr = 19,
  errCode12VOutOfLimits = 20, errCode12VVIDEOReadErr = 21, errCode12V-

VIDEOOutOfLimits = 22, errCode5VSTBYReadErr = 23,
errCode5VSTBYOutOfLimits = 24, errCode5VReadErr = 25, errCode5VOut-
OfLimits = 26, errCode3V3ReadErr = 27,
errCode3V3OutOfLimits = 28, errCodeTFTVOLReadErr = 29, errCodeTFTV-
OLOutOfLimits = 30, errCode1V9ReadErr = 31,
errCode1V9OutOfLimits = 32, errCode1V8ReadErr = 33, errCode1V8OutOf-
Limits = 34, errCode1V5ReadErr = 35,
errCode1V5OutOfLimits = 36, errCode1V2ReadErr = 37, errCode1V2OutOf-
Limits = 38, errCode1V05ReadErr = 39,
errCode1V05OutOfLimits = 40, errCode1V0ReadErr = 41, errCode1V0OutOf-
Limits = 42, errCode0V9ReadErr = 43,
errCode0V9OutOfLimits = 44, errCodeI2CTEMPReadErr = 45, errCodeI2CT-
EMPOutOfLimits = 46, errCodeSTM32TEMPReadErr = 47,
errCodeSTM32TEMPOutOfLimits = 48, errCodeBLTYPEUnexpEEPROMVal
= 49, errCodeFPBTNUnexpEEPROMVal = 50, errCodeEXTCTRLUnexpEEP-
ROMVal = 51,
errCodeLowRange24VUnexpEEPROMVal = 52, errCodeSuspToRAMUnexpE-
EPROMVal = 53, errCodeCANPWRUnexpEEPROMVal = 54, errCodeVID1P-
WRUnexpEEPROMVal = 55,
errCodeVID2PWRUnexpEEPROMVal = 56, errCodeVID3PWRUnexpEEPRO-
MVal = 57, errCodeVID4PWRUnexpEEPROMVal = 58, errCodeEXTFANUnexp-
EEPROMVal = 59,
errCodeLEDUnexpEEPROMVal = 60, errCodeUnitTypeUnexpEEPROMVal =
61, errCodeBLTYPEReadErrEEPROM = 62, errCodeFPBTNReadErrEEPRO-
M = 63,
errCodeEXTCTRLReadErrEEPROM = 64, errCodeMaxSuspTimeReadErrEE-
PROM = 65, errCodeLowRange24VReadErrEEPROM = 66, errCodeSuspToR-
AMReadErrEEPROM = 67,
errCodeCANPWRReadErrEEPROM = 68, errCodeVID1PWRReadErrEEPRO-
M = 69, errCodeVID2PWRReadErrEEPROM = 70, errCodeVID3PWRRead-
ErrEEPROM = 71,
errCodeVID4PWRReadErrEEPROM = 72, errCodeEXTFANReadErrEEPRO-
M = 73, errCodeLEDReadErrEEPROM = 74, errCodeUnitTypeReadErrEEPR-
OM = 75,
errCodeRCCInit = 76, errCodeDriverInit = 77, errCodeSetSUPPLYRESET =
78, errCodeRelSUPPLYRESET = 79,
errCodeSetSYSRESET = 80, errCodeRelSYSRESET = 81, errCodeSetPWRB-
TN = 82, errCodeRelPWRBTN = 83,
errCodeOnBL = 84, errCodeOffBL = 85, errCodeEXTFANOn = 86, errCodeE-
XTFANOff = 87,
errCodePWRENOn = 88, errCodePWRENOff = 89, errCodeMPPWRENOn =
90, errCodeMPPWRENOff = 91,
errCodeCANPWRENOn = 92, errCodeCANPWRENOff = 93, errCodeVID1P-
WRENOn = 94, errCodeVID1PWRENOff = 95,
errCodeVID2PWRENOn = 96, errCodeVID2PWRENOff = 97, errCodeVID3P-
WRENOn = 98, errCodeVID3PWRENOff = 99,
errCodeVID4PWRENOn = 100, errCodeVID4PWRENOff = 101, errCodeHE-
ATACTOn = 102, errCodeHEATACTOff = 103,
errCodeSetLEDCol = 104, errCodeSetLEDFreq = 105, errCodeManageLED =

106, errCodeManageCANPwr = 107,
errCodeManageMPPwr = 108, errCodeManageVidPwr = 109, errCodeManagePowSup = 110, errCodeManageReset = 111,
errCodeSSState = 112, errCodeVarWrapAround = 113, errCodeFPBTNUnexpVal = 114, errCodeEXTCTRLUnexpVal = 115,
errCodeMAINPWROKReadErr = 116, errCodeFRONTSPAREReadErr = 117,
errCodeTIMERReadErr = 118, errCodeManageDiagnostics = 119,
errCodeFPBTNTimOutReadErrEEPROM = 120, errCodeEXTCTRLTimOutReadErrEEPROM = 121, errCodeFPBTNAndExtCtrlDisabled = 122, errCodeSWVerReadErr = 123,
errCodeSWVerWriteErr = 124, errCodeManageActDeAct = 125, errCodeTickTimeOutTimer = 126, errCodeOperateModeStateError = 127,
errCodeHeatingTempReadErrEEPROM = 128, errCodeMPFailedStart = 129, errCodeReadErrEEPROM = 130, errCodeTimeOutWaitingForVoltages = 131,
errCodeMAX }

- enum PowerMgrConf { Normal = 0, ApplicationControlled = 1, BatterySuspend = 2 }
- enum PowerMgrStatus { NoRequestsPending = 0, SuspendPending = 1, ShutdownPending = 2 }
- enum TouchScreenModeSettings { MOUSE_NEXT_BOOT = 0, TOUCH_NEXT_BOOT = 1, MOUSE_NOW = 2, TOUCH_NOW = 3 }
- enum TSAdvancedSettingsParameter {
TS_RIGHT_CLICK_TIME = 0, TS_LOW_LEVEL = 1, TS_UNTOUCHLEVEL = 2, TS_DEBOUNCE_TIME = 3,
TS_DEBOUNCE_TIMEOUT_TIME = 4, TS_DOUBLECLICK_MAX_CLICK_TIME = 5, TS_DOUBLE_CLICK_TIME = 6, TS_MAX_RIGHTCLICK_DISTANCE = 7,
TS_USE_DEJITTER = 8, TS_CALIBTATION_WIDTH = 9, TS_CALIBRATION_MEASUREMENTS = 10, TS_RESTORE_DEFAULT_SETTINGS = 11 }
- enum CalibrationModeSettings {
MODE_UNKNOWN = 0, MODE_NORMAL = 1, MODE_CALIBRATION_5P = 2, MODE_CALIBRATION_9P = 3,
MODE_CALIBRATION_13P = 4 }
- enum CalibrationConfigParam {
CONFIG_CALIBRATION_WITH = 0, CONFIG_CALIBRATION_MEASUREMENTS = 1, CONFIG_5P_CALIBRATION_POINT_BORDER = 2, CONFIG_13P_CALIBRATION_POINT_BORDER = 3,
CONFIG_13P_CALIBRATION_TRANSITION_MIN = 4, CONFIG_13P_CALIBRATION_TRANSITION_MAX = 5 }

Functions

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

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

Variables

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

5.1.1 Typedef Documentation

5.1.1.1 typedef enum CrossControl::PowerMgrConf _PowerMgrConf

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

5.1.1.2 typedef enum CrossControl::PowerMgrStatus _PowerMgrStatus

5.1.1.3 typedef struct version_info VersionType

5.1.2 Enumeration Type Documentation

5.1.2.1 enum ButtonPowerTransitionStatus

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

Enumerator

BPTS_No_Change No change

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

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

BPTS_Restart Not currently in use

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

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

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

5.1.2.2 enum CalibrationConfigParam

Touch screen caibration parameters

Enumerator

CONFIG_CALIBRATION_WITH

CONFIG_CALIBRATION_MEASUREMENTS Accepted error value when calibrating.

CONFIG_5P_CALIBRATION_POINT_BORDER Number of measurements to accept a calibration point.

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

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

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

5.1.2.3 enum CalibrationModeSettings

Touch screen caibration modes

Enumerator

MODE_UNKNOWN

MODE_NORMAL Unknown mode.

MODE_CALIBRATION_5P Normal operation mode.

MODE_CALIBRATION_9P Calibration with 5 points mode.

MODE_CALIBRATION_13P Calibration with 9 points mode.

5.1.2.4 enum CanFrameType

Can frame type settings

Enumerator

FrameStandard

FrameExtended

FrameStandardExtended

5.1.2.5 enum CCAuxColor

Enumeration of standard colors

Enumerator

RED

GREEN RGB 0xF, 0x0, 0x0

BLUE RGB 0x0, 0xF, 0x0

CYAN RGB 0x0, 0x0, 0xF

MAGENTA RGB 0x0, 0xF, 0xF

YELLOW RGB 0xF, 0x0, 0xF

UNDEFINED_COLOR RGB 0xF, 0xF, 0x0
Returns if color is not a standard color

5.1.2.6 enum CCStatus

Enable/disable enumeration

Enumerator

Disabled

Enabled The setting is disabled or turned off

5.1.2.7 enum ChargingStatus

Current charging status of the battery.

Enumerator

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

ChargingStatus_Charging The battery is currently being charged

ChargingStatus_FullyCharged The battery is fully 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

5.1.2.8 enum DeInterlaceMode

Enumerator

DeInterlace_Even

DeInterlace_Odd Use only even rows from the interlaced input stream

DeInterlace_BOB Use only odd rows from the interlaced input stream

5.1.2.9 enum eErr

Error code enumeration

Enumerator

ERR_SUCCESS

ERR_OPEN_FAILED Success

ERR_NOT_SUPPORTED Open failed

ERR_UNKNOWN_FEATURE Not supported

ERR_DATATYPE_MISMATCH Unknown feature

ERR_CODE_NOT_EXIST Datatype mismatch

ERR_BUFFER_SIZE Code doesn't exist

ERR_IOCTRL_FAILED Buffer size error

ERR_INVALID_DATA IoCtrl operation failed

ERR_INVALID_PARAMETER Invalid data

ERR_CREATE_THREAD Invalid parameter

ERR_IN_PROGRESS Failed to create thread

ERR_CHECKSUM Operation in progress

ERR_INIT_FAILED Checksum error

ERR_VERIFY_FAILED Initalization failed

ERR_DEVICE_READ_DATA_FAILED Failed to verify

ERR_DEVICE_WRITE_DATA_FAILED Failed to read from device

ERR_COMMAND_FAILED Failed to write to device

ERR_EEPROM Command failed

ERR_JIDA_TEMP Error in EEPROM memory

ERR_AVERAGE_CALC_STARTED Failed to get JIDA temperature

ERR_NOT_RUNNING Calculation already started

ERR_I2C_EXPANDER_READ_FAILED Thread isn't running

ERR_I2C_EXPANDER_WRITE_FAILED I2C read failure

ERR_I2C_EXPANDER_INIT_FAILED I2C write failure

ERR_NEWER_SS_VERSION_REQUIRED I2C initialization failure

ERR_NEWER_FPGA_VERSION_REQUIRED SS version too old

ERR_NEWER_FRONT_VERSION_REQUIRED FPGA version too old

ERR_TELEMATICS_GPRS_NOT_AVAILABLE FRONT version too old

ERR_TELEMATICS_WLAN_NOT_AVAILABLE GPRS module not available

ERR_TELEMATICS_BT_NOT_AVAILABLE WLAN module not available

ERR_TELEMATICS_GPS_NOT_AVAILABLE Bluetooth module not available

ERR_MEM_ALLOC_FAIL GPS module not available

5.1.2.10 enum ErrorStatus

Enumerator

ErrorStatus_NoError

ErrorStatus_ThermistorTempSensor

ErrorStatus_SecondaryTempSensor

ErrorStatus_ChargeFail

ErrorStatus_Overcurrent

ErrorStatus_Init

5.1.2.11 enum hwErrorStatusCodes

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

Enumerator

errCodeNoErr

errCodeFPGACONFReadErr

errCodeFPGACONFUnexpVal

errCodeCBRESETReadErr

errCodeSUS3ReadErr

errCodeSUS4ReadErr

errCodeSUS5ReadErr

errCodePG5VSTBYReadErr

errCodePG5VSTBYUnexpVal
errCodeCANPWROKReadErr
errCodeVIDPWROKReadErr
errCodeLVDSBLENReadErr
errCodeLVDSVDDENReadErr
errCodeEXTCTRLONReadErr
errCodeFPBTNONReadErr
errCode24VReadErr
errCode24VOutOfLimits
errCode24VINReadErr
errCode24VINOutOfLimits
errCode12VReadErr
errCode12VOutOfLimits
errCode12VVIDEOReadErr
errCode12VVIDEOOutOfLimits
errCode5VSTBYReadErr
errCode5VSTBYOutOfLimits
errCode5VReadErr
errCode5VOutOfLimits
errCode3V3ReadErr
errCode3V3OutOfLimits
errCodeTFTVOLReadErr
errCodeTFTVOLOutOfLimits
errCode1V9ReadErr
errCode1V9OutOfLimits
errCode1V8ReadErr
errCode1V8OutOfLimits
errCode1V5ReadErr
errCode1V5OutOfLimits
errCode1V2ReadErr
errCode1V2OutOfLimits
errCode1V05ReadErr
errCode1V05OutOfLimits
errCode1V0ReadErr
errCode1V0OutOfLimits
errCode0V9ReadErr
errCode0V9OutOfLimits
errCodeI2CTEMPReadErr

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

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

errCodeFPBTNAndExtCtrlDisabled
errCodeSWVerReadErr
errCodeSWVerWriteErr
errCodeManageActDeAct
errCodeTickTimeOutTimer
errCodeOperateModeStateError
errCodeHeatingTempReadErrEEPROM
errCodeMPFailedStart
errCodeReadErrEEPROM
errCodeTimeOutWaitingForVoltages
errCodeMAX

5.1.2.12 enum JidaSensorType

Jida temperature sensor types

Enumerator

TEMP_CPU
TEMP_BOX
TEMP_ENV
TEMP_BOARD
TEMP_BACKPLANE
TEMP_CHIPSETS
TEMP_VIDEO
TEMP_OTHER

5.1.2.13 enum LightSensorOperationRange

Light sensor operation ranges.

Enumerator

RangeStandard
RangeExtended Light sensor operation range standard

5.1.2.14 enum LightSensorSamplingMode

Light sensor sampling modes.

Enumerator

SamplingModeStandard

SamplingModeExtended Standard sampling mode.

SamplingModeAuto Extended sampling mode.

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

5.1.2.15 enum PowerAction

Button and on/off signal actions.

Enumerator

NoAction No action taken

ActionSuspend The system enters suspend mode

ActionShutDown The system shuts down

5.1.2.16 enum PowerMgrConf

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

Enumerator

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

ApplicationControlled Applications can delay suspend/shutdown requests.

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

5.1.2.17 enum PowerMgrStatus

Enumerator

NoRequestsPending No suspend or shutdown requests.

SuspendPending A suspend request is pending.

ShutdownPending A shutdown request is pending.

5.1.2.18 enum PowerSource

Current power source of the computer.

Enumerator

PowerSource_Battery

PowerSource_ExternalPower

5.1.2.19 enum shutDownReasonCodes

The shutdown codes returned by getShutDownReason.

Enumerator

shutdownReasonCodeNoError

5.1.2.20 enum startupReasonCodes

The restart codes returned by getStartupReason.

Enumerator

startupReasonCodeUndefined

startupReasonCodeButtonPress Unknown startup reason.

startupReasonCodeExtCtrl The system was started by front panel button press

startupReasonCodeMPRestart The system was started by the external control signal

startupReasonCodePowerOnStartup The system was restarted by OS request

5.1.2.21 enum TouchScreenModeSettings

Touch screen USB profile settings

Enumerator

MOUSE_NEXT_BOOT

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

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

TOUCH_NOW Immediately set the touch USB profile to mouse profile.

5.1.2.22 enum TriggerConf

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

Enumerator

Front_Button_Enabled Front button is enabled for startup and wake-up

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

Both_Button_And_Signal_Enabled Both of the above are enabled

5.1.2.23 enum TSAdvancedSettingsParameter

Touch screen advanced settings parameters

Enumerator

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

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

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

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

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

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

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

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

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

TS_CALIBTATION_WIDTH Accepted difference in measurement during calibration of a point.

TS_CALIBRATION_MEASUREMENTS Number of measurements needed to accept a calibration point.

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

5.1.2.24 enum UpgradeAction

Upgrade Action enumeration

Enumerator

UPGRADE_INIT

UPGRADE_PREP_COM Initiating, checking for compatibility etc

UPGRADE_READING_FILE Preparing communication

UPGRADE_CONVERTING_FILE Opening and reading the supplied file

UPGRADE_FLASHING Converting the mcs format to binary format

UPGRADE_VERIFYING Flashing the file

UPGRADE_COMPLETE Verifying the programmed image

UPGRADE_COMPLETE_WITH_ERRORS Upgrade was finished
Upgrade finished prematurely, see errorCode for the reason of failure

5.1.2.25 enum VideoChannel

The available analog video channels

Enumerator

Analog_Channel_1

Analog_Channel_2

Analog_Channel_3

Analog_Channel_4

5.1.2.26 enum videoStandard

Enumerator

STD_M_J_NTSC

STD_B_D_G_H_I_N_PAL (M,J) NTSC ITU-R BT6.01

STD_M_PAL (B, D, G, H, I, N) PAL ITU-R BT6.01

STD_PAL (M) PAL ITU-R BT6.01

STD_NTSC PAL-Nc ITU-R BT6.01

STD_SECAM NTSC 4.43 ITU-R BT6.01

5.1.2.27 enum VoltageEnum

Voltage type enumeration

Enumerator

VOLTAGE_24VIN

VOLTAGE_24V $<$ 24VIN

VOLTAGE_12V $<$ 24V

VOLTAGE_12VID $<$ 12V

VOLTAGE_5V $<$ 12VID

VOLTAGE_3V3 $<$ 5V

VOLTAGE_VTFT $<$ 3.3V

VOLTAGE_5VSTB $<$ VTFT
VOLTAGE_1V9 $<$ 5VSTB
VOLTAGE_1V8 $<$ 1.9V
VOLTAGE_1V5 $<$ 1.8V
VOLTAGE_1V2 $<$ 1.5V
VOLTAGE_1V05 $<$ 1.2V
VOLTAGE_1V0 $<$ 1.05V
VOLTAGE_0V9 $<$ 1.0V
VOLTAGE_VREF_INT $<$ 0.9V
$<$ SS internal VRef

5.1.3 Function Documentation

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

CCAux Error handling

Get a string description of an error code.

Parameters

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

Returns

String description of an error code.

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

Get a string description of an error code.

Parameters

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

Returns

String description of an error code.

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

String access functions for codes used in the diagnostic functions

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

Parameters

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

Returns

String description of an error code.

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

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

Parameters

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

Returns

String description of an error code.

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

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

Parameters

code	A code for which to get a string description.

Returns

String description of a code.

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

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

Parameters

code	A code for which to get a string description.

Returns

String description of a code.

5.1.4 Variable Documentation

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

Bit defines for getDigIO

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

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

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

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

Bit defines for getVideoStartupPowerConfig and setVideoStartupPowerConfig

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

Video channel 1 config

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

Video channel 2 config

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

Video channel 3 config

Chapter 6

Data Structure Documentation

6.1 About Struct Reference

`#include <About.h>`

Public Member Functions

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

6.1.1 Detailed Description

Get information about the CCpilot XM computer.

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

Example Usage:

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

using namespace std;

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

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

  if(!buffer)
    return;

  CrossControl::eErr err;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  delete[] buffer;
}


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

  list_about_information(pAbout);

  pAbout->Release();
}
```

6.1.2 Member Function Documentation

6.1.2.1 virtual eErr getAddOnHWversion (char * *buff*, int *length*) [pure virtual]

Get Add on hardware version.

Parameters

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.

Example Usage:

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

6.1.2.2 virtual eErr getAddOnManufacturingDate (char * *buff*, int *length*) [pure virtual]

Get Add on manufacturing date.

Parameters

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.

Example Usage:

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

6.1.2.3 virtual eErr getAddOnPCBArt (char * *buff*, int *length*) [pure virtual]

Get Add on PCB article number.

Parameters

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.

Example Usage:

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

6.1.2.4 virtual eErr getAddOnPCBSerial (char * *buff*, int *length*) [pure virtual]

Get Add on PCB serial number.

Parameters

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.

Example Usage:

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

6.1.2.5 virtual eErr getIsBTMounted (bool * *mounted*) [pure virtual]

Get BlueTooth module mounting status.

Parameters

mounted	Is module mounted?

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

available	Is display available?

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.7 virtual eErr getIsGPRSMounted (bool * *mounted*) [pure virtual]

Get GPRS module mounting status.

Parameters

mounted	Is module mounted?

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.8 virtual eErr getIsGPSMounted (bool * *mounted*) [pure virtual]

Get GPS module mounting status.

Parameters

mounted	Is module mounted?

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get Display TouchScreen status.

Parameters

available	Is TouchScreen available?

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.10 virtual eErr getIsWLANMounted (bool * *mounted*) [pure virtual]

Get WLAN module mounting status.

Parameters

mounted	Is module mounted?

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
bool isWLANMounted;
```

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

6.1.2.11 virtual eErr getMainHWversion (char * *buff*, int *len*) [pure virtual]

Get main hardware version (PCB revision).

Parameters

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.

Example Usage:

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

6.1.2.12 virtual eErr getMainManufacturingDate (char * *buff*, int *len*) [pure virtual]

Get main manufacturing date.

Parameters

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.

Example Usage:

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

6.1.2.13 virtual eErr getMainPCBArt (char * *buff*, int *length*) [pure virtual]

Get main PCB article number.

Parameters

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.

Example Usage:

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

6.1.2.14 virtual eErr getMainPCBSerial (char * *buff*, int *len*) [pure virtual]

Get main PCB serial number.

Parameters

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.

Example Usage:

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

6.1.2.15 virtual eErr getMainProdArtNr (char * *buff*, int *length*) [pure virtual]

Get main product article number.

Parameters

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.

Example Usage:

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

6.1.2.16 virtual eErr getMainProdRev (char * *buff*, int *len*) [pure virtual]

Get main product revision.

Parameters

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.

Example Usage:

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

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

Get number of CAN connections present.

Parameters

NrOf-Connections	Returns the number of connections.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get number of digital I/O connections present.

Parameters

NrOfConnections	Returns the number of input or input/output connections.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.19 virtual eErr getNrOfETHConnections (unsigned char * *NrOfConnections*) [pure virtual]

Get number of ethernet connections present.

Parameters

NrOfConnections	Returns the number of connections.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.20 virtual eErr getNrOfSerialConnections (unsigned char * *NrOfConnections*) [pure virtual]

Get number of serial port (RS232) connections present.

Parameters

NrOfConnections	Returns the number of connections.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.21 virtual eErr getNrOfUSBConnections (unsigned char * *NrOfConnections*) [pure virtual]

Get number of USB connections present.

Parameters

NrOf-Connections	Returns the number of connections.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.1.2.22 virtual eErr getNrOfVideoConnections (unsigned char * *NrOfConnections*) [pure virtual]

Get number of Video connections present.

Parameters

NrOf-Connections	Returns the number of connections.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

```
unsigned char nrOfVideoConnections;
```

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

6.1.2.23 virtual eErr getUnitSerial (char * *buff*, int *len*) [pure virtual]

Get unit serial number.

Parameters

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.

Example Usage:

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

6.1.2.24 virtual void Release ( ) [pure virtual]

Delete the About object.

Returns

-

Example Usage:

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

list_about_information(pAbout);

pAbout->Release();
```

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

- IncludeFiles/About.h

6.2 Adc Struct Reference

`#include <Adc.h>`

Public Member Functions

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

6.2.1 Detailed Description

Get current voltages from the built-in ADC

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

Example Usage:

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

using namespace std;

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

  CrossControl::eErr err;
  double voltage;

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


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

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

  pAdc->Release();
}
```

6.2.2 Member Function Documentation

6.2.2.1 virtual CrossControl::eErr getVoltage (VoltageEnum *selection*, double * *value*) [pure virtual]

Read measured voltage.

Parameters

selection	The type of voltage to get.
value	Voltage value in Volt.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.2.2.2 virtual void Release ( ) [pure virtual]

Delete the ADC object.

Returns

-

Example Usage:

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

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

pAdc->Release();
```

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

- IncludeFiles/Adc.h

6.3 AuxVersion Struct Reference

`#include <AuxVersion.h>`

Public Member Functions

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

6.3.1 Detailed Description

Get software versions for firmware and software

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

Example Usage:

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

using namespace std;

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

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

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

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

```
else
  cout << "unknown" << endl;

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

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

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

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

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

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

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

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

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

cout << setw(column_width) << "CCAux Driver Version: ";
if (CrossControl::ERR_SUCCESS != err)
  cout << (int) major << "." <<
       (int) minor << "." <<
```

```
        (int) release << "." <<
        (int) build << endl;
  else
    cout << "unknown" << endl;
}

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

  output_versions(pAuxVersion);


  pAuxVersion->Release();
}
```

6.3.2 Member Function Documentation

6.3.2.1 virtual eErr getCCAuxDrvVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

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

Parameters

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.

Example Usage:

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

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

6.3.2.2 virtual eErr getCCAuxVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the CrossControl CCAux API version. CCAux includes: CCAuxDrv - Hardware driver. CCAuxService - Windows Service. ccauxd - Linux daemon. CCAuxDll - The

dll implementing this API.

Parameters

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.

Example Usage:

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

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

6.3.2.3 virtual eErr getFPGAVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the FPGA software version

Parameters

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.

Example Usage:

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

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

6.3.2.4 virtual eErr getFrontVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the front microcontroller software version

Parameters

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.

Example Usage:

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

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

6.3.2.5 virtual eErr getOSVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the CrossControl Operating System version.

Parameters

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.

Example Usage:

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

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

6.3.2.6 virtual eErr getSSVersion (unsigned char * *major*, unsigned char * *minor*, unsigned char * *release*, unsigned char * *build*) [pure virtual]

Get the System Supervisor software version

Parameters

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.

Example Usage:

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

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

6.3.2.7 virtual void Release ( ) [pure virtual]

Delete the AuxVersion object.

Returns

-

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

- IncludeFiles/AuxVersion.h

6.4 Backlight Struct Reference

`#include <Backlight.h>`

Public Member Functions

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

6.4.1 Detailed Description

Backlight settings

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

Example Usage:

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

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

  CrossControl::eErr err;
  unsigned char value;

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

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

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

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

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

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



}


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

  change_backlight(pBacklight);

  pBacklight->Release();
}
```

6.4.2 Member Function Documentation

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

Get light sensor filter parameters for automatic backlight control.

Parameters

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.

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

Get parameters for automatic backlight control.

Parameters

bSoftTransitions	Soft transitions used?
k	K value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.4.2.3 virtual eErr getAutomaticBLStatus (unsigned char * *status*) [pure virtual]

Get status from automatic backlight control.

Parameters

status	1=running, 0=stopped.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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

Parameters

intensity	The current backlight intensity (3..255).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

6.4.2.5 virtual eErr getLedDimming (CCStatus * *status*) [pure virtual]

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

Parameters

status	Enabled/Disabled

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get backlight controller status.

Parameters

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.

Example Usage:

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

6.4.2.7 virtual void Release ( ) [pure virtual]

Delete the backlight object.

Returns

-

Example Usage:

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

change_backlight(pBacklight);

pBacklight->Release();
```

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

Set light sensor filter parameters for automatic backlight control.

Parameters

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.

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

Set parameters for automatic backlight control.

Parameters

bSoftTransitions	Use soft transitions?

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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

Parameters

intensity	The backlight intensity to set (3..255).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

6.4.2.11 virtual eErr setLedDimming (CCStatus *status*) [pure virtual]

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

Parameters

status	Enabled/Disabled

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.4.2.12 virtual eErr startAutomaticBL ( ) [pure virtual]

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.4.2.13 virtual eErr stopAutomaticBL ( ) [pure virtual]

Stop automatic backlight control.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

- IncludeFiles/Backlight.h

6.5 Battery Struct Reference

`#include <Battery.h>`

Public Member Functions

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

6.5.1 Detailed Description

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

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

Example Usage:

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

using namespace std;
using namespace CrossControl;


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

  if(!pBattery)
    return;

  error = pBattery->isBatteryPresent(&bpresent);

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

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

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


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

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


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


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

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



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


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

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


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

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


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


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

}


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

  readBatteryInfo(pBattery);

  pBattery->Release();
```

```
}
```

6.5.2 Member Function Documentation

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

Get battery charging status.

Parameters

status	the current charging mode of the battery.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get battery hardware version (PCB revision).

Parameters

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.

Example Usage:

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

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

Get battery serial number.

Parameters

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.

Example Usage:

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

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

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

Get the battery software version

Parameters

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.

Example Usage:

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

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

Get battery temperature.

Parameters

temperature	PCB Temperature in degrees Celsius.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get battery voltage status.

Parameters

batteryVoltagePercent	the current voltage level of the battery, in percent [0..100].

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

errorCode	Error code. Zero means no error.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Get temperature interval of the battery.

Parameters

minTemp	Minimum measured temperature.
maxTemp	Maximum measured temperature.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get the currently used power source.

Parameters

status	the current power source, external power or battery.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get battery diagnostic timer.

Parameters

times	Get a struct with the current diagnostic times.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Is an external battery connected?

Parameters

batteryIs-Present	true if a battery is connected, otherwise false.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

6.5.2.12 virtual void Release ( ) [pure virtual]

Delete the Battery object.

Returns

-

Example Usage:

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

readBatteryInfo(pBattery);

pBattery->Release();
```

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

- IncludeFiles/Battery.h

6.6 BatteryTimerType Struct Reference

`#include <Battery.h>`

Data Fields

- unsigned long TotRunTimeMain
- unsigned long TotRunTimeBattery
- unsigned long RunTime_m20
- unsigned long RunTime_m20_0
- unsigned long RunTime_0_40
- unsigned long RunTime_40_60
- unsigned long RunTime_60_70
- unsigned long RunTime_70_80
- unsigned long RunTime_Above80

6.6.1 Detailed Description

Diagnostic timer data

6.6.2 Field Documentation

6.6.2.1 unsigned long RunTime_0_40

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

6.6.2.2 unsigned long RunTime_40_60

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

6.6.2.3 unsigned long RunTime_60_70

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

6.6.2.4 unsigned long RunTime_70_80

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

6.6.2.5 unsigned long RunTime_Above80

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

6.6.2.6 unsigned long RunTime_m20

Total running time on battery power (minutes)

6.6.2.7 unsigned long RunTime_m20_0

Total runtime below -20 deg C (minutes)

6.6.2.8 unsigned long TotRunTimeBattery

Total running time on main power (minutes)

6.6.2.9 unsigned long TotRunTimeMain

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

- IncludeFiles/Battery.h

6.7 Buzzer Struct Reference

`#include <Buzzer.h>`

Public Member Functions

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

6.7.1 Detailed Description

Buzzer settings

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

Example Usage:

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

using namespace std;
using namespace CrossControl;

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

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


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

```
    return;

  CrossControl::eErr err;
  unsigned short vol;

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

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

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

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

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

  play_beeps(pBuzzer);

  pBuzzer->Release();
}
```

6.7.2 Member Function Documentation

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

Buzzes for a specified time.

Parameters

time	Time (ms) to buzz.
blocking	Blocking or non-blocking function.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

6.7.2.2 virtual eErr getFrequency (unsigned short * *frequency*) [pure virtual]

Get buzzer frequency.

Parameters

frequency	Current frequency (700-10000 Hz).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.7.2.3 virtual eErr getTrigger (bool * *trigger*) [pure virtual]

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

Parameters

trigger	Current trigger status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get buzzer volume.

Parameters

volume	Current volume (0-51).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.7.2.5 virtual void Release ( ) [pure virtual]

Delete the Buzzer object.

Returns

-

Example Usage:

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

play_beeps(pBuzzer);

pBuzzer->Release();
```

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

Set buzzer frequency.

Parameters

frequency	Frequency to set (700-10000 Hz).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.7.2.7 virtual eErr setTrigger (bool *trigger*) [pure virtual]

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

Parameters

trigger	Status to set.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Set buzzer volume.

Parameters

volume	Volume to set (0-51).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

- IncludeFiles/Buzzer.h

6.8 BuzzerSetup Struct Reference

`#include <CCAuxTypes.h>`

Data Fields

- unsigned short frequency
- unsigned short volume

6.8.1 Field Documentation

6.8.1.1 unsigned short frequency

buzzer frequency

6.8.1.2 unsigned short volume

buzzer volume

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

- IncludeFiles/CCAuxTypes.h

6.9 CanSetting Struct Reference

`#include <CanSetting.h>`

Public Member Functions

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

6.9.1 Detailed Description

Can settings

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

Example Usage:

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

using namespace std;
using namespace CrossControl;


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

```
  if(!pCanSetting)
    return;

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

#ifndef LINUX

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

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


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

  read_cansettings(pCanSetting);

  pCanSetting->Release();
}
```

6.9.2 Member Function Documentation

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

Get Baud rate

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get frame type

Parameters

net	CAN net (1-4) to get settings for.
frameType	CAN frame type

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.9.2.3 virtual void Release ( ) [pure virtual]

Delete the CanSetting object.

Returns

-

Example Usage:

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

read_cansettings(pCanSetting);

pCanSetting->Release();
```

6.9.2.4 virtual eErr setBaudrate (unsigned char *net*, unsigned short *baudrate*) [pure virtual]

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

Parameters

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.

6.9.2.5 virtual eErr setFrameType (unsigned char *net*, CanFrameType *frameType*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

- IncludeFiles/CanSetting.h

6.10 Config Struct Reference

`#include <Config.h>`

Public Member Functions

- virtual eErr getStartupTriggerConfig (TriggerConf *configuration)=0
- virtual eErr getShortButtonPressAction (PowerAction *action)=0
- virtual eErr getLongButtonPressAction (PowerAction *action)=0
- virtual eErr getOnOffSigAction (PowerAction *action)=0
- virtual eErr getFrontBtnTrigTime (unsigned short *triggertime)=0

- virtual eErr getExtOnOffSigTrigTime (unsigned long *triggertime)=0
- virtual eErr getSuspendMaxTime (unsigned short *maxTime)=0
- virtual eErr getCanStartupPowerConfig (CCStatus *status)=0
- virtual eErr getVideoStartupPowerConfig (unsigned char *config)=0
- virtual eErr getExtFanStartupPowerConfig (CCStatus *status)=0
- virtual eErr getStartupVoltageConfig (double *voltage)=0
- virtual eErr getHeatingTempLimit (signed short *temperature)=0
- virtual eErr getPowerOnStartup (CCStatus *status)=0
- virtual eErr setStartupTriggerConfig (TriggerConf conf)=0
- virtual eErr setShortButtonPressAction (PowerAction action)=0
- virtual eErr setLongButtonPressAction (PowerAction action)=0
- virtual eErr setOnOffSigAction (PowerAction action)=0
- virtual eErr setFrontBtnTrigTime (unsigned short triggertime)=0
- virtual eErr setExtOnOffSigTrigTime (unsigned long triggertime)=0
- virtual eErr setSuspendMaxTime (unsigned short maxTime)=0
- virtual eErr setCanStartupPowerConfig (CCStatus status)=0
- virtual eErr setVideoStartupPowerConfig (unsigned char config)=0
- virtual eErr setExtFanStartupPowerConfig (CCStatus status)=0
- virtual eErr setStartupVoltageConfig (double voltage)=0
- virtual eErr setHeatingTempLimit (signed short temperature)=0
- virtual eErr setPowerOnStartup (CCStatus status)=0
- virtual void Release ()=0

6.10.1 Detailed Description

Video channel 4 config

Configuration of various settings

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

Example Usage:

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

using namespace std;
using namespace CrossControl;


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

  if(!pConfig)
    return;

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

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

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


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

  conf_example(pConfig);

  pConfig->Release();
}
```

6.10.2 Member Function Documentation

6.10.2.1 virtual eErr getCanStartupPowerConfig (CCStatus * *status*) [pure virtual]

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

Parameters

status	Enabled/Disabled

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.2 virtual eErr getExtFanStartupPowerConfig (CCStatus * *status*) [pure virtual]

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

Parameters

status	Enabled/Disabled

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.3 virtual eErr getExtOnOffSigTrigTime (unsigned long * *triggertime*) [pure virtual]

Get external on/off signal trigger time.

Parameters

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.

6.10.2.4 virtual eErr getFrontBtnTrigTime (unsigned short * *triggertime*) [pure virtual]

Get front button trigger time for long press.

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.

6.10.2.5 virtual eErr getHeatingTempLimit (signed short * *temperature*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.6 virtual eErr getLongButtonPressAction (PowerAction * *action*) [pure virtual]

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

Parameters

action	The configured action.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.7 virtual eErr getOnOffSigAction (PowerAction * *action*) [pure virtual]

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

Parameters

action	The configured action.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.8 virtual eErr getPowerOnStartup (CCStatus * *status*) [pure virtual]

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

Parameters

status	Enabled/Disabled

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.9 virtual eErr getShortButtonPressAction (PowerAction * *action*) [pure virtual]

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

Parameters

action	The configured action.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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

Parameters

configuration	One of: Front_Button_Enabled, OnOff_Signal_Enabled or Both_Button_And_Signal_Enabled.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.10.2.11 virtual eErr getStartupVoltageConfig (double * *voltage*) [pure virtual]

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

Parameters

voltage	The current voltage setting. (9V .. 28V)

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.12 virtual eErr getSuspendMaxTime (unsigned short * *maxTime*) [pure virtual]

Get suspend mode maximum time.

Parameters

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.

6.10.2.13 virtual eErr getVideoStartupPowerConfig (unsigned char * *config*) [pure virtual]

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

Parameters

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.

6.10.2.14 virtual void Release ( ) [pure virtual]

Delete the Config object.

Returns

-

Example Usage:

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

conf_example(pConfig);

pConfig->Release();
```

6.10.2.15 virtual eErr setCanStartupPowerConfig (CCStatus *status*) [pure virtual]

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

Parameters

status	Enabled/Disabled

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.16 virtual eErr setExtFanStartupPowerConfig (CCStatus *status*) [pure virtual]

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

Parameters

status	Enabled/Disabled

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.17 virtual eErr setExtOnOffSigTrigTime (unsigned long *triggertime*) [pure virtual]

Set external on/off signal trigger time.

Parameters

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.

6.10.2.18 virtual eErr setFrontBtnTrigTime (unsigned short *triggertime*) [pure virtual]

Set front button trigger time for long press.

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.

6.10.2.19 virtual eErr setHeatingTempLimit (signed short *temperature*) [pure virtual]

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

Parameters

temperature	The heating limit, in degrees Celsius (-25 to +5)

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.20 virtual eErr setLongButtonPressAction (PowerAction *action*) [pure virtual]

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

Parameters

action	The action to set.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.21 virtual eErr setOnOffSigAction (PowerAction *action*) [pure virtual]

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

Parameters

action	The action to set.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.22 virtual eErr setPowerOnStartup (CCStatus *status*) [pure virtual]

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

Parameters

status	Enabled/Disabled

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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

Parameters

action	The action to set.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.10.2.24 virtual eErr setStartupTriggerConfig (TriggerConf *conf*) [pure virtual]

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

Parameters

conf	Must be one of: Front_Button_Enabled, OnOff_Signal_Enabled or Both_Button_And_Signal_Enabled.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.25 virtual eErr setStartupVoltageConfig (double *voltage*) [pure virtual]

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

Parameters

voltage	The voltage to set (9V .. 28V).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.10.2.26 virtual eErr setSuspendMaxTime (unsigned short *maxTime*) [pure virtual]

Set suspend mode maximum time.

Parameters

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.

6.10.2.27 virtual eErr setVideoStartupPowerConfig (unsigned char *config*) [pure virtual]

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

Parameters

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.

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

- IncludeFiles/Config.h

6.11 Diagnostic Struct Reference

`#include <Diagnostic.h>`

Public Member Functions

- virtual eErr getSSTemp (signed short ∗temperature)=0
- virtual eErr getPCBTemp (signed short ∗temperature)=0
- virtual eErr getPMTemp (unsigned char index, signed short ∗temperature, JidaSensorType ∗jst)=0
- virtual eErr getStartupReason (unsigned short ∗reason)=0
- virtual eErr getShutDownReason (unsigned short ∗reason)=0
- virtual eErr getHwErrorStatus (unsigned short ∗errorCode)=0
- virtual eErr getTimer (TimerType ∗times)=0
- virtual eErr getMinMaxTemp (signed short ∗minTemp, signed short ∗maxTemp)=0
- virtual eErr getPowerCycles (unsigned short ∗powerCycles)=0
- virtual eErr clearHwErrorStatus (void)=0
- virtual void Release ()=0

6.11.1 Detailed Description

Access to unit diagnostic data

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

Example Usage:

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

using namespace std;
using namespace CrossControl;


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

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

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

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

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

  if(!pDiagnostic)
    return;

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

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

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

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

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

  diagnostic_example(pDiagnostic);

  pDiagnostic->Release();
}
```

6.11.2 Member Function Documentation

6.11.2.1 virtual eErr clearHwErrorStatus (void) [pure virtual]

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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

Parameters

errorCode	Error code. Zero means no error.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get diagnostic temperature interval of the unit.

Parameters

minTemp	Minimum measured PCB temperature.
maxTemp	Maximum measured PCB temperature.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.11.2.4 virtual eErr getPCBTemp (signed short * *temperature*) [pure virtual]

Get PCB temperature.

Parameters

temperature	PCB Temperature in degrees Celsius.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.11.2.5 virtual eErr getPMTemp (unsigned char *index*, signed short * *temperature*, JidaSensorType * *jst*) [pure virtual]

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.11.2.6 virtual eErr getPowerCycles (unsigned short * *powerCycles*) [pure virtual]

Get number of power cycles.

Parameters

powerCycles	Total number of power cycles.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.11.2.7 virtual eErr getShutDownReason (unsigned short * *reason*) [pure virtual]

Get shutdown reason.

Parameters

reason	See DiagnosticCodes.h for shutdown codes.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get System Supervisor temperature.

Parameters

temperature	System Supervisor temperature in degrees Celsius.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.11.2.9 virtual eErr getStartupReason (unsigned short * *reason*) [pure virtual]

Get startup reason.

Parameters

reason	See DiagnosticCodes.h for startup codes.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get diagnostic timer.

Parameters

times	Get a struct with the current diagnostic times.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.11.2.11 virtual void Release ( ) [pure virtual]

Delete the Diagnostic object.

Returns

-

Example Usage:

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

diagnostic_example(pDiagnostic);

pDiagnostic->Release();
```

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

- IncludeFiles/Diagnostic.h

6.12 DigIO Struct Reference

`#include <DigIO.h>`

Public Member Functions

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

6.12.1 Detailed Description

Read digital inputs

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

Example Usage:

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

using namespace std;

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

  CrossControl::eErr err;
  unsigned char inputs;

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


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

  list_digital_inputs(pDigIO);

  pDigIO->Release();
}
```

6.12.2 Member Function Documentation

6.12.2.1 virtual eErr getDigIO (unsigned char * *status*) [pure virtual]

Get Digital inputs.

Parameters

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.

Example Usage:

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

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

Get Digital io direction.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

6.12.2.3 virtual void Release ( ) [pure virtual]

Delete the DigIO object.

Returns

-

Example Usage:

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

list_digital_inputs(pDigIO);

pDigIO->Release();
```

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

Set Digital outputs.

Parameters

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 are always zero.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

Set Digital io direction.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

- IncludeFiles/DigIO.h

6.13 FirmwareUpgrade Struct Reference

`#include <FirmwareUpgrade.h>`

Public Member Functions

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

6.13.1 Detailed Description

Firmware upgrade of the system's microprocessors and FPGA.

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

Example Usage:

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

using namespace std;

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

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

  cout << "Upgrading SS" << endl;

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

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

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

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

  upgrade->Release();
}

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

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

  cout << "Upgrading front" << endl;

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

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

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

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

  upgrade->Release();
```

```
}

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

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

  cout << "Upgrading FPGA" << endl;

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

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

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

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

  upgrade->Release();
}

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

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

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

  return 0;
}
```

6.13.2 Member Function Documentation

6.13.2.1 virtual eErr getUpgradeStatus (UpgradeStatus * *status*, bool *blocking*) [pure virtual]

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

Parameters

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.

6.13.2.2 virtual void Release ( ) [pure virtual]

Delete the FirmwareUpgrade object.

Returns

-

Example Usage:

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

6.13.2.3 virtual eErr shutDown ( ) [pure virtual]

Shut down the operating system.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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

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

Parameters

filename	Path and filename to the .mcs 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 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.

Example Usage:

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

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

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

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

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

6.13.2.5 virtual eErr startFpgaVerification (const char * *filename*, bool *blocking*) [pure virtual]

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

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

Parameters

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.

Example Usage:

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

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

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

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

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

6.13.2.6 virtual eErr startFrontUpgrade (const char * *filename*, bool *blocking*) [pure virtual]

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

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

Parameters

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.

Example Usage:

```
cout << "Upgrading front" << endl;

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

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

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

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

6.13.2.7 virtual eErr startFrontVerification (const char * *filename*, bool *blocking*) [pure virtual]

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

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

Parameters

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.

Example Usage:

```
cout << "Upgrading front" << endl;

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

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

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

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

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

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

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

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.

Example Usage:

```
cout << "Upgrading SS" << endl;

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

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

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

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

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

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

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

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.

Example Usage:

```
cout << "Upgrading SS" << endl;

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

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

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

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

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

- IncludeFiles/FirmwareUpgrade.h

6.14 FrontLED Struct Reference

`#include <FrontLED.h>`

Public Member Functions

- virtual eErr getSignal (double *frequency, unsigned char *dutyCycle)=0
- virtual eErr getOnTime (unsigned char *onTime)=0

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

6.14.1 Detailed Description

Front LED control

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

Example Usage:

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

using namespace std;
using namespace CrossControl;

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

  if(!pFrontLED)
    return;

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

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

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

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


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

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

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

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

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


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

  //
  // Restore settings
  //

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

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

}


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

  led_example(pFrontLED);

  pFrontLED->Release();
}
```

6.14.2 Member Function Documentation

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

Get front LED color mix.

Parameters

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.

Example Usage:

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

6.14.2.2 virtual eErr getColor (CCAuxColor * *color*) [pure virtual]

Get front LED color.

Parameters

color	Color from CCAuxColor

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.3 virtual eErr getEnabledDuringStartup (CCStatus * *status*) [pure virtual]

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

Parameters

status	LED Enabled or Disabled during startup.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.4 virtual eErr getIdleTime (unsigned char * *idleTime*) [pure virtual]

Get front LED idle time.

Parameters

idleTime	Time in 100ms.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.5 virtual eErr getNrOfPulses (unsigned char * *nrOfPulses*) [pure virtual]

Get number of pulses during a blink sequence.

Parameters

nrOfPulses	Number of pulses.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.6 virtual eErr getOffTime (unsigned char * *offTime*) [pure virtual]

Get front LED off time.

Parameters

offTime	Time in 10ms.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.7 virtual eErr getOnTime (unsigned char * *onTime*) [pure virtual]

Get front LED on time.

Parameters

onTime	Time in 10ms. 0 = off

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.14.2.9 virtual void Release ( ) [pure virtual]

Delete the FrontLED object.

Returns

-

Example Usage:

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

led_example(pFrontLED);

pFrontLED->Release();
```

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

Set front LED color mix.

Parameters

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.

Example Usage:

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

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

Set one of the front LED standard colors.

Parameters

color	Color from CCAuxColor

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.14.2.12 virtual eErr setEnabledDuringStartup (CCStatus *status*) [pure virtual]

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

Parameters

status	Enable or Disable the LED during startup.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.13 virtual eErr setIdleTime (unsigned char *idleTime*) [pure virtual]

Get front LED idle time.

Parameters

idleTime	Time in 100ms.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.14 virtual eErr setNrOfPulses (unsigned char *nrOfPulses*) [pure virtual]

Set front LED number of pulses during a blink sequence.

Parameters

nrOfPulses	Number of pulses.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.14.2.15 virtual eErr setOff ( ) [pure virtual]

Set front LED off.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Set front LED off time.

Parameters

offTime	Time in 10ms.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Set front LED on time.

Parameters

onTime	Time in 10ms. 0 = off

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Set front LED signal.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

- IncludeFiles/FrontLED.h

6.15 LedColorMixType Struct Reference

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

Data Fields

- unsigned char red
- unsigned char green
- unsigned char blue

6.15.1 Field Documentation

6.15.1.1 unsigned char blue

Blue color intensity 0-0x0F

6.15.1.2 unsigned char green

Green color intensity 0-0x0F

6.15.1.3 unsigned char red

Red color intensity 0-0x0F

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

- IncludeFiles/CCAuxTypes.h

6.16 LedTimingType Struct Reference

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

Data Fields

- unsigned char onTime
- unsigned char offTime
- unsigned char idleTime
- unsigned char nrOfPulses

6.16.1 Field Documentation

6.16.1.1 unsigned char idleTime

LED idle time in 100ms

6.16.1.2 unsigned char nrOfPulses

Pulses per sequences

6.16.1.3 unsigned char offTime

LED off time in 10ms

6.16.1.4 unsigned char onTime

LED on time in 10ms

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

- IncludeFiles/CCAuxTypes.h

6.17 Lightsensor Struct Reference

`#include <Lightsensor.h>`

Public Member Functions

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

6.17.1 Detailed Description

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

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

Example Usage:

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

```
using namespace std;
using namespace CrossControl;

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

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

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

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

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

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

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

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

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


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

  ls_example(pLightSensor);

  pLightSensor->Release();
}
```

6.17.2 Member Function Documentation

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

Get average illuminance (light) value from light sensor.

Parameters

value	Illuminance value (Lux).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.17.2.2 virtual eErr getIlluminance (unsigned short * *value*) [pure virtual]

Get illuminance (light) value from light sensor.

Parameters

value	Illuminace value (Lux).

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.17.2.3 virtual eErr getIlluminance (unsigned short * *value*, unsigned char * *ch0*, unsigned char * *ch1*) [pure virtual]

Get illuminance (light) value from light sensor.

Parameters

value	Illuminance value (Lux).
ch0	Channel0 value.
ch1	Channel1 value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.17.2.4 virtual eErr getOperatingRange (LightSensorOperationRange * *range*) [pure virtual]

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

Parameters

range	Operating range. RangeStandard or RangeExtended.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.17.2.5 virtual void Release ( ) [pure virtual]

Delete the Lightsensor object.

Returns

-

Example Usage:

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

ls_example(pLightSensor);

pLightSensor->Release();
```

6.17.2.6 virtual eErr setOperatingRange (LightSensorOperationRange *range*) [pure virtual]

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

Parameters

range	Operating range to set. RangeStandard or RangeExtended.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Start average calculation.

Parameters

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.

Example Usage:

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

6.17.2.8 virtual eErr stopAverageCalc ( ) [pure virtual]

Stop average calculation.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

- IncludeFiles/Lightsensor.h

6.18 Power Struct Reference

`#include <Power.h>`

Public Member Functions

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

6.18.1 Detailed Description

Power control access functions

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

Example Usage:

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

using namespace std;
using namespace CrossControl;

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

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


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


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

}


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

  power_example(pPower);

  pPower->Release();
}
```

6.18.2 Member Function Documentation

6.18.2.1 virtual eErr ackPowerRequest ( ) [pure virtual]

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get backlight power status.

Parameters

status	Backlight power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.18.2.3 virtual eErr getButtonPowerTransitionStatus (ButtonPowerTransitionStatus * *status*) [pure virtual]

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

Parameters

status	The current status. See the definition of ButtonPowerTransitionStatus for details.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.18.2.4 virtual eErr getCanPowerStatus (CCStatus * *status*) [pure virtual]

Get can power status.

Parameters

status	Can power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.18.2.5 virtual eErr getExtFanPowerStatus (CCStatus * *status*) [pure virtual]

Get external fan power status.

Parameters

status	Fan power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get Video power status.

Parameters

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.

Example Usage:

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

6.18.2.7 virtual void Release ( ) [pure virtual]

Delete the Power object.

Returns

-

Example Usage:

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

power_example(pPower);

pPower->Release();
```

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

Set backlight power status.

Parameters

status	Backlight power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.18.2.9 virtual eErr setCanPowerStatus (CCStatus *status*) [pure virtual]

Set can power status.

Parameters

status	Can power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.18.2.10 virtual eErr setExtFanPowerStatus (CCStatus *status*) [pure virtual]

Set external fan power status.

Parameters

status	Fan power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.18.2.11 virtual eErr setVideoPowerStatus (unsigned char *status*) [pure virtual]

Set Video power status.

Parameters

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.

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

- IncludeFiles/Power.h

6.19 PowerMgr Struct Reference

`#include <PowerMgr.h>`

Public Member Functions

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

6.19.1 Detailed Description

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

Example Usage:

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

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



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

  CrossControl::eErr err;
  PowerMgrConf setConfiguration;

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

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



while(1)
{
  OSSleep(500);

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

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

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

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

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

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

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

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

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

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

        // Expecting to get configuration Normal after resume from suspend

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


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

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

int main(void)
{
```

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

  assert(pPowerMgr);
  assert(pBattery);

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

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

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

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

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

  test_powermgr(pPowerMgr);

  pPowerMgr->Release();
}
```

6.19.2 Member Function Documentation

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

Get the configuration that is in use.

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 = pPowerMgr->getConfiguration(&conf);
if(err == ERR_SUCCESS)
{
  switch (conf)
```

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

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

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

Parameters

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

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

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

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

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

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

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

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

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

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

  // Expecting to get configuration Normal after resume from suspend

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

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

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

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

Returns

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

Example Usage:

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

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

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

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

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

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

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

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

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

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

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

  // Expecting to get configuration Normal after resume from suspend

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


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

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

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

Configure the PowerMgr. Call this function once initially to turn on the functionality.

Parameters

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

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

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

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

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

test_powermgr(pPowerMgr);

pPowerMgr->Release();
```

6.19.2.5 virtual void Release ( ) [pure virtual]

Delete the PowerMgr object.

Returns

-

Example Usage:

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

assert(pPowerMgr);
assert(pBattery);

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

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

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

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

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

test_powermgr(pPowerMgr);

pPowerMgr->Release();
```

6.19.2.6 virtual eErr setAppReadyForSuspendOrShutdown (void) [pure virtual]

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

acknowledge a request within 20s from when it arrives.

Returns

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

Example Usage:

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

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

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

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

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

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

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

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

```
    }

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

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

      // Expecting to get configuration Normal after resume from suspend

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


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

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

- IncludeFiles/PowerMgr.h

6.20 received_video Struct Reference

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

Data Fields

- unsigned short received_width
- unsigned short received_height
- unsigned char received_framerate

6.20.1 Field Documentation

6.20.1.1 unsigned char received_framerate

6.20.1.2 unsigned short received_height

6.20.1.3 unsigned short received_width

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

- IncludeFiles/CCAuxTypes.h

6.21 Smart Struct Reference

`#include <Smart.h>`

Public Member Functions

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

6.21.1 Detailed Description

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

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

Example Usage:

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

using namespace std;

void show_card_data(SMARTHANDLE pSmart)
{
```

```
  if(!pSmart)
    return;

  CrossControl::eErr err;

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


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

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

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


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

  show_card_data(pSmart);

  pSmart->Release();
}
```

6.21.2 Member Function Documentation

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

Get serial number of the secondary storage device.

Parameters

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.

Example Usage:

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

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

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

Parameters

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.

Example Usage:

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

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

Get remaining lifetime of the secondary storage device.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.21.2.4 virtual void Release ( ) [pure virtual]

Delete the Smart object.

Returns

-

Example Usage:

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

show_card_data(pSmart);

pSmart->Release();
```

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

- IncludeFiles/Smart.h

6.22 Telematics Struct Reference

`#include <Telematics.h>`

Public Member Functions

- virtual eErr getTelematicsAvailable (CCStatus *status)=0
- virtual eErr getGPRSPowerStatus (CCStatus *status)=0
- virtual eErr getGPRSStartUpPowerStatus (CCStatus *status)=0
- virtual eErr getWLANPowerStatus (CCStatus *status)=0
- virtual eErr getWLANStartUpPowerStatus (CCStatus *status)=0
- virtual eErr getGPSAntennaStatus (CCStatus *status)=0
- virtual eErr setGPRSPowerStatus (CCStatus status)=0
- virtual eErr setGPRSStartUpPowerStatus (CCStatus status)=0
- virtual eErr setWLANPowerStatus (CCStatus status)=0
- virtual eErr setWLANStartUpPowerStatus (CCStatus status)=0
- virtual void Release ()=0
- virtual eErr getBTPowerStatus (CCStatus *status)=0
- virtual eErr getBTStartUpPowerStatus (CCStatus *status)=0
- virtual eErr getGPSPowerStatus (CCStatus *status)=0
- virtual eErr getGPSStartUpPowerStatus (CCStatus *status)=0
- virtual eErr setBTPowerStatus (CCStatus status)=0
- virtual eErr setBTStartUpPowerStatus (CCStatus status)=0
- virtual eErr setGPSPowerStatus (CCStatus status)=0
- virtual eErr setGPSStartUpPowerStatus (CCStatus status)=0

6.22.1 Detailed Description

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

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

Example Usage:

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

using namespace std;
using namespace CrossControl;

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


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

```
    GetErrorStringA(err) << endl;
  return;
}

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

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


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

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


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

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

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

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

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

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

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

  telematics_example(pTelematics);

  pTelematics->Release();
}
```

6.22.2 Member Function Documentation

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

Get Bluetooth power status.

Parameters

status	Bluetooth power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

status	Bluetooth power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get GPRS power status.

Parameters

status	GPRS power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

status	GPRS power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

status	GPS antenna power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

status	GPS power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

status	GPS power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Is a telematics add-on card installed?

Parameters

status	Enabled if a telematics add-on card is installed, otherwise Disabled.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

Get WLAN power status.

Parameters

status	WLAN power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

status	WLAN power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.22.2.11 virtual void Release ( ) [pure virtual]

Delete the Telematics object.

Returns

-

Example Usage:

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

telematics_example(pTelematics);

pTelematics->Release();
```

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

Set Bluetooth power status.

Parameters

status	Bluetooth power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.13 virtual eErr setBTStartUpPowerStatus (CCStatus *status*) [pure virtual]

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

Parameters

status	Bluetooth power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.14 virtual eErr setGPRSPowerStatus (CCStatus *status*) [pure virtual]

Set GPRS modem power status.

Parameters

status	GPRS modem power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.15 virtual eErr setGPRSStartUpPowerStatus (CCStatus *status*) [pure virtual]

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

Parameters

status	GPRS power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Set GPS power status.

Parameters

status	GPS power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

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

Parameters

status	GPS power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.18 virtual eErr setWLANPowerStatus (CCStatus *status*) [pure virtual]

Set WLAN power status.

Parameters

status	WLAN power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.22.2.19 virtual eErr setWLANStartUpPowerStatus (CCStatus *status*) [pure virtual]

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

Parameters

status	WLAN power status.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

- IncludeFiles/Telematics.h

6.23 TimerType Struct Reference

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

Data Fields

- unsigned long TotRunTime
- unsigned long TotSuspTime
- unsigned long TotHeatTime
- unsigned long RunTime40_60
- unsigned long RunTime60_70
- unsigned long RunTime70_80
- unsigned long Above80RunTime

6.23.1 Detailed Description

Diagnostic timer data

6.23.2 Field Documentation

6.23.2.1 unsigned long Above80RunTime

Total runtime in 70-80deg (minutes)

6.23.2.2 unsigned long RunTime40_60

Total heating time (minutes)

6.23.2.3 unsigned long RunTime60_70

Total runtime in 40-60deg (minutes)

6.23.2.4 unsigned long RunTime70_80

Total runtime in 60-70deg (minutes)

6.23.2.5 unsigned long TotHeatTime

Total suspend time (minutes)

6.23.2.6 unsigned long TotRunTime

6.23.2.7 unsigned long TotSuspTime

Total running time (minutes)

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

- IncludeFiles/CCAuxTypes.h

6.24 TouchScreen Struct Reference

`#include <TouchScreen.h>`

Public Member Functions

- virtual eErr getMode (TouchScreenModeSettings *config)=0
- virtual eErr getMouseRightClickTime (unsigned short *time)=0
- virtual eErr setMode (TouchScreenModeSettings config)=0
- virtual eErr setMouseRightClickTime (unsigned short time)=0
- virtual eErr setAdvancedSetting (TSAdvancedSettingsParameter param, unsigned short data)=0
- virtual eErr getAdvancedSetting (TSAdvancedSettingsParameter param, unsigned short *data)=0
- virtual void Release ()=0

6.24.1 Detailed Description

Touch Screen settings

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

Example Usage:

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

using namespace std;
using namespace CrossControl;

void touchscreen_example(TOUCHSCREENHANDLE pTouchScreen)
{
```

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

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

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


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


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

  touchscreen_example(pTouchScreen);

  pTouchScreen->Release();
}
```

6.24.2 Member Function Documentation

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

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

Parameters

param	The setting to get.
data	The current data for the setting.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

config	The current mode.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

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

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

Parameters

time	The right click time, in milliseconds.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

Example Usage:

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

6.24.2.4 virtual void Release ( ) [pure virtual]

Delete the TouchScreen object.

Returns

-

Example Usage:

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

touchscreen_example(pTouchScreen);

pTouchScreen->Release();
```

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

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

Parameters

param	The setting to set.
data	The data value to set.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.24.2.6 virtual eErr setMode (TouchScreenModeSettings *config*) [pure virtual]

Set Touch Screen mode. Sets the mode of the USB profile.

Parameters

config	The mode to set.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.24.2.7 virtual eErr setMouseRightClickTime (unsigned short *time*) [pure virtual]

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

Parameters

time	The right click time, in milliseconds.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

- IncludeFiles/TouchScreen.h

6.25 TouchScreenCalib Struct Reference

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

Public Member Functions

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

6.25.1 Detailed Description

Touch Screen Calibration interface

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

6.25.2 Member Function Documentation

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

Check if a calibration point is finished

Parameters

finished	Is current point finished?
pointNr	Calibration point number (1 to total number of points)

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get calibration config parameters

Parameters

param	Config parameter
value	Parameter value

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Get mode of front controller.

Parameters

mode	Current calibration mode

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.25.2.4 virtual void Release ( ) [pure virtual]

Delete the TouchScreenCalib object.

Returns

-

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

Set calibration point

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Set calibration config parameters

Parameters

param	Config parameter
value	parameter value

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Set mode of front controller.

Parameters

mode	Selected calibration mode

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

- IncludeFiles/TouchScreenCalib.h

6.26 UpgradeStatus Struct Reference

`#include <CCAuxTypes.h>`

Data Fields

- enum UpgradeAction currentAction
- unsigned char percent
- eErr errorCode

6.26.1 Detailed Description

Upgrade Status

6.26.2 Field Documentation

6.26.2.1 enum UpgradeAction currentAction

6.26.2.2 eErr errorCode

Represents the percentage of completion of the current action

6.26.2.3 unsigned char percent

The current action.

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

- IncludeFiles/CCAuxTypes.h

6.27 version_info Struct Reference

`#include <CCAuxTypes.h>`

Data Fields

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

6.27.1 Field Documentation

6.27.1.1 unsigned char build

version build number

6.27.1.2 unsigned char major

version major number

6.27.1.3 unsigned char minor

version minor number

6.27.1.4 unsigned char release

version release number

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

- IncludeFiles/CCAuxTypes.h

6.28 Video Struct Reference

`#include <Video.h>`

Public Member Functions

- virtual eErr init (unsigned char deviceNr)=0
- virtual eErr showVideo (bool show)=0
- virtual eErr setDeInterlaceMode (DeInterlaceMode mode)=0
- virtual eErr getDeInterlaceMode (DeInterlaceMode *mode)=0
- virtual eErr setMirroring (CCStatus mode)=0
- virtual eErr getMirroring (CCStatus *mode)=0
- virtual eErr setActiveChannel (VideoChannel channel)=0
- virtual eErr getActiveChannel (VideoChannel *channel)=0
- virtual eErr setColorKeys (unsigned char rKey, unsigned char gKey, unsigned char bKey)=0
- virtual eErr getColorKeys (unsigned char *rKey, unsigned char *gKey, unsigned char *bKey)=0
- virtual eErr setVideoArea (unsigned short topLeftX, unsigned short topLeftY, unsigned short bottomRigthX, unsigned short bottomRigthY)=0
- virtual eErr getVideoArea (unsigned short *topLeftX, unsigned short *topLeftY, unsigned short *bottomRightX, unsigned short *bottomRightY)=0
- virtual eErr getRawImage (unsigned short *width, unsigned short *height, float *frameRate)=0
- virtual eErr getVideoStandard (videoStandard *standard)=0
- virtual eErr getStatus (unsigned char *status)=0
- virtual eErr setScaling (float x, float y)=0
- virtual eErr getScaling (float *x, float *y)=0
- virtual eErr activateSnapshot (bool activate)=0
- virtual eErr takeSnapshot (const char *path, bool bInterlaced)=0
- virtual eErr takeSnapshotRaw (char *rawImgBuffer, unsigned long rawImgBuffSize, bool bInterlaced)=0
- virtual eErr takeSnapshotBmp (char **bmpBuffer, unsigned long *bmpBufSize, bool bInterlaced, bool bNTSCFormat)=0
- virtual eErr createBitmap (char **bmpBuffer, unsigned long *bmpBufSize, const char *rawImgBuffer, unsigned long rawImgBufSize, bool bInterlaced, bool bNTSCFormat)=0
- virtual eErr freeBmpBuffer (char *bmpBuffer)=0
- virtual eErr minimize ()=0
- virtual eErr restore ()=0
- virtual eErr setDecoderReg (unsigned char decoderRegister, unsigned char registerValue)=0
- virtual eErr getDecoderReg (unsigned char decoderRegister, unsigned char *registerValue)=0
- virtual eErr setCropping (unsigned char top, unsigned char left, unsigned char bottom, unsigned char right)=0
- virtual eErr getCropping (unsigned char *top, unsigned char *left, unsigned char *bottom, unsigned char *right)=0
- virtual void Release ()=0

6.28.1 Detailed Description

Analog Video

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

6.28.2 Member Function Documentation

6.28.2.1 virtual eErr activateSnapshot (bool *activate*) [pure virtual]

To be able to take snapshot the snapshot function has to be active. After activation it takes 120ms before first snapshot can be taken. The Snapshot function can be active all the time. If power consumption and heat is an issue, snapshot may be turned off.

Parameters

activate	Set to true if the snapshot function shall be active.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.2 virtual eErr createBitmap (char ** *bmpBuffer*, unsigned long * *bmpBufSize*, const char * *rawImgBuffer*, unsigned long *rawImgBufSize*, bool *bInterlaced*, bool *bNTSCFormat*) [pure virtual]

Create a bitmap from a raw image buffer. The bmp buffer is allocated in the function and has to be deallocated by the application.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

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

Free the memory allocated for BMP buffer.

Parameters

bmpBuffer	The bmp buffer to free.

Returns

error status.

6.28.2.4 virtual eErr getActiveChannel (VideoChannel * *channel*) [pure virtual]

Get the current video channel.

Parameters

channel	Enum defining available channels.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.5 virtual eErr getColorKeys (unsigned char * *rKey,* unsigned char * *gKey,* unsigned char * *bKey*) [pure virtual]

Get color key values. Note that the system uses 18 bit colors, so the two least significant bits are not used.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.6 virtual eErr getCropping (unsigned char * *top,* unsigned char * *left,* unsigned char * *bottom,* unsigned char * *right*) [pure virtual]

Get Crop parameters.

Parameters

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.

6.28.2.7 virtual eErr getDecoderReg (unsigned char *decoderRegister,* unsigned char * *registerValue*) [pure virtual]

Get the Video decoder bus register. Advanced function for direct access to the video decoder TVP5150AM1 registers.

Parameters

decoder-Register	Decoder Register Address.
register-Value	register value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.8 virtual eErr getDeInterlaceMode (DeInterlaceMode * *mode*) [pure virtual]

Get the deinterlace mode used when decoding the interlaced video stream.

Parameters

mode	The current mode. See enum DeInterlaceMode for descriptions of the modes.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.9 virtual eErr getMirroring (CCStatus * *mode*) [pure virtual]

Get the current mirroring mode of the video image.

Parameters

mode	The current mode. Enabled or Disabled.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.10 virtual eErr getRawImage (unsigned short * *width*, unsigned short * *height*, float * *frameRate*) [pure virtual]

Get the raw image size of moving image before any scaling and frame rate. For snapshot the height is 4 row less.

Parameters

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.

6.28.2.11 virtual eErr getScaling (float * *x*, float * *y*) [pure virtual]

Get Video Scaling (image size). If the deinterlace mode is set to DeInterlace_Even or DeInterlace_Odd, this function divides the actual vertical scaling by a factor of two, to get the same scaling factor as set with setScaling.

Parameters

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.

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

Video status byte.

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.

6.28.2.13 virtual eErr getVideoArea (unsigned short * *topLeftX*, unsigned short * *topLeftY*, unsigned short * *bottomRightX*, unsigned short * *bottomRightY*) [pure virtual]

Get the area where video is shown.

Parameters

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.

6.28.2.14 virtual eErr getVideoStandard (videoStandard * *standard*) [pure virtual]

Get video standard. The video decoder auto detects the video standard of the source.

Parameters

standard	Video standard.

Returns

error status.

6.28.2.15 virtual eErr init (unsigned char *deviceNr*) [pure virtual]

Initialize a video device. The video device will initially use the following settings: DeInterlace_BOB and mirroring disabled.

Parameters

deviceNr	Device to connect to (1,2). Select one of 2 devices to connect to.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.16 virtual eErr minimize ( ) [pure virtual]

Minimizes the video area. Restore with restore() call.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.17 virtual void Release ( ) [pure virtual]

Delete the Video object.

Returns

-

6.28.2.18 virtual eErr restore ( ) [pure virtual]

Restores the video area to the size it was before a minimize() call. Don't use restore if minimize has not been used first.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.19 virtual eErr setActiveChannel (VideoChannel *channel*) [pure virtual]

Sets the active video channel.

Parameters

channel	Enum defining available channels.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.20 virtual eErr setColorKeys (unsigned char *rKey,* unsigned char *gKey,* unsigned char *bKey*) [pure virtual]

Set color keys. Writes RGB color key values. Note that the system uses 18 bit colors, so the two least significant bits are not used.

Parameters

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

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.21 virtual eErr setCropping (unsigned char *top,* unsigned char *left,* unsigned char *bottom,* unsigned char *right*) [pure virtual]

Crop video image. Note that the video chip manual says the following about 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.

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.

6.28.2.22 virtual eErr setDecoderReg (unsigned char *decoderRegister*, unsigned char *registerValue*) [pure virtual]

Set the Video decoder bus register. Advanced function for direct access to the video decoder TVP5150AM1 registers.

Parameters

decoder-Register	Decoder Register Address.
register-Value	register value.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.23 virtual eErr setDeInterlaceMode (DeInterlaceMode *mode*) [pure virtual]

Set the deinterlace mode used when decoding the interlaced video stream.

Parameters

mode	The mode to set. See enum DeInterlaceMode for descriptions of the modes.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.24 virtual eErr setMirroring (CCStatus *mode*) [pure virtual]

Enable or disable mirroring of the video image.

Parameters

mode	The mode to set. Enabled or Disabled.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.25 virtual eErr setScaling (float *x*, float *y*) [pure virtual]

Set Video Scaling (image size). If the deinterlace mode is set to DeInterlace_Even or DeInterlace_Odd, this function multiplies the vertical scaling by a factor of two, to get the correct image proportions.

Parameters

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.

6.28.2.26 virtual eErr setVideoArea (unsigned short *topLeftX*, unsigned short *topLeftY*, unsigned short *bottomRigthX*, unsigned short *bottomRigthY*) [pure virtual]

Set the area where video is shown.

Parameters

topLeftX	Top left X coordinate on screen.
topLeftY	Top left Y coordinate on screen.
bottom-RigthX	Bottom right X coordinate on screen.
bottom-RigthY	Bottom right Y coordinate on screen.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.27 virtual eErr showVideo (bool *show*) [pure virtual]

Show or hide the video image. Note that it may take some time before the video is shown and correct input info can be read by getRawImage.

Parameters

show	True shows the video image.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.28 virtual eErr takeSnapshot (const char * *path*, bool *bInterlaced*) [pure virtual]

Takes a snapshot of the current video image and stores it to a bitmap file. This is a combination of takeSnapShotRaw, getVideoStandard and createBitMap and then storing of the bmpBuffer to file. To be able to take a snapshot, the snapshot function has to be active.

Parameters

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.

6.28.2.29 virtual eErr takeSnapshotBmp (char ** *bmpBuffer*, unsigned long * *bmpBufSize*, bool *bInterlaced*, bool *bNTSCFormat*) [pure virtual]

Takes a snapshot of the current video image and return a data buffer with a bitmap image. The bmp buffer is allocated in the function and has to be deallocated with freeBmpBuffer() by the application. This is a combination of the function takeSnapShotRaw and createBitMap. To be able to take a snapshot, the snapshot function has to be active.

Parameters

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.
bNTSC-Format	True if the video format in rawImageBuffer is NTSC format.

Returns

error status. 0 = ERR_SUCCESS, otherwise error code.

6.28.2.30 virtual eErr takeSnapshotRaw (char * *rawImgBuffer,* unsigned long *rawImgBuffSize,* bool *bInterlaced*) [pure virtual]

Takes a snapshot of the current video image and return raw image data. The size of the raw image is when interlaced = false 0x100 + line count * row count * 4. The size of the raw image is when interlaced = true 0x100 + line count * row count * 2. To be able to take a snapshot, the snapshot function has to be active. This function is blocking until a new frame is available from the decoder. An error will be returned if the decoder doesn't return any frames before a timeout.

Parameters

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.

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

- IncludeFiles/Video.h

6.29 video_dec_command Struct Reference

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

Data Fields

- unsigned char decoder_register
- unsigned char register_value

6.29.1 Field Documentation

6.29.1.1 unsigned char decoder_register

6.29.1.2 unsigned char register_value

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

- IncludeFiles/CCAuxTypes.h

Chapter 7

File Documentation

7.1 IncludeFiles/About.h File Reference

Data Structures

- struct About

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::About * ABOUTHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  ABOUTHANDLE
  CCAUXDLL_CALLING_CONV GetAbout (void)

7.1.1 Typedef Documentation

7.1.1.1 typedef CrossControl::About* ABOUTHANDLE

7.1.2 Function Documentation

7.1.2.1 EXTERN_C CCAUXDLL_API ABOUTHANDLE CCAUXDLL_CALLING_CONV GetAbout (void)

Factory function that creates instances of the About object.

Returns

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

Example Usage:

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

list_about_information(pAbout);

pAbout->Release();
```

7.2 IncludeFiles/Adc.h File Reference

Data Structures

- struct Adc

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Adc * ADCHANDLE

Enumerations

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

Functions

- EXTERN_C CCAUXDLL_API
  ADCHANDLE
  CCAUXDLL_CALLING_CONV GetAdc (void)

7.2.1 Typedef Documentation

7.2.1.1 typedef CrossControl::Adc* ADCHANDLE

7.2.2 Function Documentation

7.2.2.1 EXTERN_C CCAUXDLL_API ADCHANDLE CCAUXDLL_CALLING_CONV GetAdc (void)

Factory function that creates instances of the Adc object.

Returns

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

Example Usage:

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

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

pAdc->Release();
```

7.3 IncludeFiles/AuxVersion.h File Reference

Data Structures

- struct AuxVersion

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::AuxVersion * AUXVERSIONHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  AUXVERSIONHANDLE
  CCAUXDLL_CALLING_CONV GetAuxVersion (void)

7.3.1 Typedef Documentation

7.3.1.1 typedef CrossControl::AuxVersion* AUXVERSIONHANDLE

7.3.2 Function Documentation

7.3.2.1 EXTERN_C CCAUXDLL_API AUXVERSIONHANDLE CCAUXDLL_CALLING_CONV GetAuxVersion (void)

Factory function that creates instances of the AuxVersion object.

Returns

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

Example Usage:

```
AUXVERSIONHANDLE pAuxVersion = ::GetAuxVersion();
assert (pAuxVersion);

output_versions(pAuxVersion);


pAuxVersion->Release();
```

7.4 IncludeFiles/Backlight.h File Reference

Data Structures

- struct Backlight

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Backlight * BACKLIGHTHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  BACKLIGHTHANDLE
  CCAUXDLL_CALLING_CONV GetBacklight (void)

7.4.1 Typedef Documentation

7.4.1.1 typedef CrossControl::Backlight* BACKLIGHTHANDLE

7.4.2 Function Documentation

7.4.2.1 EXTERN_C CCAUXDLL_API BACKLIGHTHANDLE CCAUXDLL_CALLING_CONV GetBacklight (void)

Factory function that creates instances of the Backlight object.

Returns

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

Example Usage:

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

change_backlight(pBacklight);

pBacklight->Release();
```

7.5 IncludeFiles/Battery.h File Reference

Data Structures

- struct BatteryTimerType
- struct Battery

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Battery * BATTERYHANDLE

Enumerations

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

Functions

- EXTERN_C CCAUXDLL_API
  BATTERYHANDLE
  CCAUXDLL_CALLING_CONV GetBattery (void)

7.5.1 Typedef Documentation

7.5.1.1 typedef CrossControl::Battery* BATTERYHANDLE

7.5.2 Function Documentation

7.5.2.1 EXTERN_C CCAUXDLL_API BATTERYHANDLE CCAUXDLL_CALLING_CONV GetBattery (void)

Factory function that creates instances of the Battery object.

Returns

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

Example Usage:

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

readBatteryInfo(pBattery);

pBattery->Release();
```

7.6 IncludeFiles/Buzzer.h File Reference

Data Structures

- struct Buzzer

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Buzzer * BUZZERHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  BUZZERHANDLE
  CCAUXDLL_CALLING_CONV GetBuzzer (void)

7.6.1 Typedef Documentation

7.6.1.1 typedef CrossControl::Buzzer* BUZZERHANDLE

7.6.2 Function Documentation

7.6.2.1 EXTERN_C CCAUXDLL_API BUZZERHANDLE CCAUXDLL_CALLING_CONV GetBuzzer (void)

Factory function that creates instances of the Buzzer object.

Returns

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

Example Usage:

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

play_beeps(pBuzzer);

pBuzzer->Release();
```

7.7 IncludeFiles/CanSetting.h File Reference

Data Structures

- struct CanSetting

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::CanSetting * CANSETTINGHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  CANSETTINGHANDLE
  CCAUXDLL_CALLING_CONV GetCanSetting (void)

7.7.1 Typedef Documentation

7.7.1.1 typedef CrossControl::CanSetting* CANSETTINGHANDLE

7.7.2 Function Documentation

7.7.2.1 EXTERN_C CCAUXDLL_API CANSETTINGHANDLE CCAUXDLL_CALLING_CONV GetCanSetting (void)

Factory function that creates instances of the CanSetting object.

Returns

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

Example Usage:

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

read_cansettings(pCanSetting);

pCanSetting->Release();
```

7.8 IncludeFiles/CCAuxErrors.h File Reference

Namespaces

- namespace CrossControl

Functions

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

7.9 IncludeFiles/CCAuxTypes.h File Reference

Data Structures

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

Namespaces

- namespace CrossControl

Typedefs

- typedef struct version_info VersionType

Enumerations

- enum LightSensorOperationRange { RangeStandard = 0, RangeExtended = 1 }
- enum LightSensorSamplingMode { SamplingModeStandard = 0, SamplingModeExtended, SamplingModeAuto }
- enum CCStatus { Disabled = 0, Enabled = 1 }
- enum eErr {
  ERR_SUCCESS = 0, ERR_OPEN_FAILED = 1, ERR_NOT_SUPPORTED = 2, ERR_UNKNOWN_FEATURE = 3,
  ERR_DATATYPE_MISMATCH = 4, ERR_CODE_NOT_EXIST = 5, ERR_BUFFER_SIZE = 6, ERR_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_FAILED = 22, ERR_I2C_EXPANDER_WRITE_FAILED = 23,
ERR_I2C_EXPANDER_INIT_FAILED = 24, ERR_NEWER_SS_VERSION_REQUIRED = 25, ERR_NEWER_FPGA_VERSION_REQUIRED = 26, ERR_NEWER_FRONT_VERSION_REQUIRED = 27,
ERR_TELEMATICS_GPRS_NOT_AVAILABLE = 28, ERR_TELEMATICS_WLAN_NOT_AVAILABLE = 29, ERR_TELEMATICS_BT_NOT_AVAILABLE = 30, ERR_TELEMATICS_GPS_NOT_AVAILABLE = 31,
ERR_MEM_ALLOC_FAIL = 32 }

- enum DeInterlaceMode { DeInterlace_Even = 0, DeInterlace_Odd = 1, DeInterlace_BOB = 2 }
- enum VideoChannel { Analog_Channel_1 = 0, Analog_Channel_2 = 1, Analog_Channel_3 = 2, Analog_Channel_4 = 3 }
- enum videoStandard {
  STD_M_J_NTSC = 0, STD_B_D_G_H_I_N_PAL = 1, STD_M_PAL = 2, STD_PAL = 3,
  STD_NTSC = 4, STD_SECAM = 5 }
- enum CanFrameType { FrameStandard, FrameExtended, FrameStandardExtended }
- enum TriggerConf { Front_Button_Enabled = 1, OnOff_Signal_Enabled = 2, Both_Button_And_Signal_Enabled = 3 }
- enum PowerAction { NoAction = 0, ActionSuspend = 1, ActionShutDown = 2 }
- enum ButtonPowerTransitionStatus {
  BPTS_No_Change = 0, BPTS_ShutDown = 1, BPTS_Suspend = 2, BPTS_Restart = 3,
  BPTS_BtnPressed = 4, BPTS_BtnPressedLong = 5, BPTS_SignalOff = 6 }
- enum JidaSensorType {
  TEMP_CPU = 0, TEMP_BOX = 1, TEMP_ENV = 2, TEMP_BOARD = 3,
  TEMP_BACKPLANE = 4, TEMP_CHIPSETS = 5, TEMP_VIDEO = 6, TEMP_OTHER = 7 }
- enum UpgradeAction {
  UPGRADE_INIT, UPGRADE_PREP_COM, UPGRADE_READING_FILE, UPGRADE_CONVERTING_FILE,
  UPGRADE_FLASHING, UPGRADE_VERIFYING, UPGRADE_COMPLETE, UPGRADE_COMPLETE_WITH_ERRORS }
- enum CCAuxColor {
  RED = 0, GREEN, BLUE, CYAN,
  MAGENTA, YELLOW, UNDEFINED_COLOR }

7.10 IncludeFiles/CCPlatform.h File Reference

7.11 IncludeFiles/Config.h File Reference

Data Structures

- struct Config

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Config * CONFIGHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  CONFIGHANDLE
  CCAUXDLL_CALLING_CONV GetConfig (void)

Variables

- const unsigned char Video1Conf = (1 << 0)
- const unsigned char Video2Conf = (1 << 1)
- const unsigned char Video3Conf = (1 << 2)
- const unsigned char Video4Conf = (1 << 3)

7.11.1 Typedef Documentation

7.11.1.1 typedef CrossControl::Config* CONFIGHANDLE

7.11.2 Function Documentation

7.11.2.1 EXTERN_C CCAUXDLL_API CONFIGHANDLE CCAUXDLL_CALLING_CONV GetConfig (void)

Factory function that creates instances of the Config object.

Returns

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

Example Usage:

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

conf_example(pConfig);

pConfig->Release();
```

7.12 IncludeFiles/Diagnostic.h File Reference

Data Structures

- struct Diagnostic

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Diagnostic * DIAGNOSTICHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  DIAGNOSTICHANDLE
  CCAUXDLL_CALLING_CONV GetDiagnostic (void)

7.12.1 Typedef Documentation

7.12.1.1 typedef CrossControl::Diagnostic* DIAGNOSTICHANDLE

7.12.2 Function Documentation

7.12.2.1 EXTERN_C CCAUXDLL_API DIAGNOSTICHANDLE CCAUXDLL_CALLING_CONV GetDiagnostic (void)

Factory function that creates instances of the Diagnostic object.

Returns

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

Example Usage:

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

diagnostic_example(pDiagnostic);

pDiagnostic->Release();
```

7.13 IncludeFiles/DiagnosticCodes.h File Reference

Namespaces

- namespace CrossControl

Enumerations

- enum startupReasonCodes {
  startupReasonCodeUndefined = 0x0000, startupReasonCodeButtonPress = 0x0055,
  startupReasonCodeExtCtrl = 0x00AA, startupReasonCodeMPRestart = 0x00F0,
  startupReasonCodePowerOnStartup = 0x000F }
- enum shutDownReasonCodes { shutdownReasonCodeNoError = 0x001F }
- enum hwErrorStatusCodes {
  errCodeNoErr = 0, errCodeFPGACONFReadErr = 1, errCodeFPGACONFUnexpVal = 2, errCodeCBRESETReadErr = 3,
  errCodeSUS3ReadErr = 4, errCodeSUS4ReadErr = 5, errCodeSUS5ReadErr = 6, errCodePG5VSTBYReadErr = 7,
  errCodePG5VSTBYUnexpVal = 8, errCodeCANPWROKReadErr = 9, errCodeVIDPWROKReadErr = 10, errCodeLVDSBLENReadErr = 11,
  errCodeLVDSVDDENReadErr = 12, errCodeEXTCTRLONReadErr = 13, errCodeFPBTNONReadErr = 14, errCode24VReadErr = 15,
  errCode24VOutOfLimits = 16, errCode24VINReadErr = 17, errCode24VINOutOfLimits = 18, errCode12VReadErr = 19,
  errCode12VOutOfLimits = 20, errCode12VVIDEOReadErr = 21, errCode12VVIDEOOutOfLimits = 22, errCode5VSTBYReadErr = 23,
  errCode5VSTBYOutOfLimits = 24, errCode5VReadErr = 25, errCode5VOutOfLimits = 26, errCode3V3ReadErr = 27,
  errCode3V3OutOfLimits = 28, errCodeTFTVOLReadErr = 29, errCodeTFTVOLOutOfLimits = 30, errCode1V9ReadErr = 31,
  errCode1V9OutOfLimits = 32, errCode1V8ReadErr = 33, errCode1V8OutOfLimits = 34, errCode1V5ReadErr = 35,
  errCode1V5OutOfLimits = 36, errCode1V2ReadErr = 37, errCode1V2OutOfLimits = 38, errCode1V05ReadErr = 39,
  errCode1V05OutOfLimits = 40, errCode1V0ReadErr = 41, errCode1V0OutOfLimits = 42, errCode0V9ReadErr = 43,
  errCode0V9OutOfLimits = 44, errCodeI2CTEMPReadErr = 45, errCodeI2CTEMPOutOfLimits = 46, errCodeSTM32TEMPReadErr = 47,
  errCodeSTM32TEMPOutOfLimits = 48, errCodeBLTYPEUnexpEEPROMVal = 49, errCodeFPBTNUnexpEEPROMVal = 50, errCodeEXTCTRLUnexpEEPROMVal = 51,
  errCodeLowRange24VUnexpEEPROMVal = 52, errCodeSuspToRAMUnexpE-

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

Functions

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

7.14 IncludeFiles/DigIO.h File Reference

Data Structures

- struct DigIO

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::DigIO * DIGIOHANDLE

Functions

- EXTERN_C CCAUXDLL_API DIGIOHANDLE CCAUXDLL_CALLING_CONV GetDigIO (void)

Variables

- const unsigned char DigitalIn_1 = (1 << 0)
- const unsigned char DigitalIn_2 = (1 << 1)
- const unsigned char DigitalIn_3 = (1 << 2)
- const unsigned char DigitalIn_4 = (1 << 3)

7.14.1 Typedef Documentation

7.14.1.1 typedef CrossControl::DigIO* DIGIOHANDLE

7.14.2 Function Documentation

7.14.2.1 EXTERN_C CCAUXDLL_API DIGIOHANDLE CCAUXDLL_CALLING_CONV GetDigIO (void)

Factory function that creates instances of the DigIO object.

Returns

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

Example Usage:

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

list_digital_inputs(pDigIO);

pDigIO->Release();
```

7.15 IncludeFiles/FirmwareUpgrade.h File Reference

Data Structures

- struct FirmwareUpgrade

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::FirmwareUpgrade * FIRMWAREUPGHANDLE

Functions

- EXTERN_C CCAUXDLL_API FIRMWAREUPGHANDLE CCAUXDLL_CALLING_CONV GetFirmwareUpgrade (void)

7.15.1 Typedef Documentation

7.15.1.1 typedef CrossControl::FirmwareUpgrade* FIRMWAREUPGHANDLE

7.15.2 Function Documentation

7.15.2.1 EXTERN_C CCAUXDLL_API FIRMWAREUPGHANDLE CCAUXDLL_CALLING_CONV GetFirmwareUpgrade (void)

Factory function that creates instances of the Adc object.

Returns

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

Example Usage:

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

7.16 IncludeFiles/FrontLED.h File Reference

Data Structures

- struct FrontLED

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::FrontLED * FRONTLEDHANDLE

Functions

- EXTERN_C CCAUXDLL_API FRONTLEDHANDLE CCAUXDLL_CALLING_CONV GetFrontLED (void)

7.16.1 Typedef Documentation

7.16.1.1 typedef CrossControl::FrontLED* FRONTLEDHANDLE

7.16.2 Function Documentation

7.16.2.1 EXTERN_C CCAUXDLL_API FRONTLEDHANDLE CCAUXDLL_CALLING_CONV GetFrontLED (void)

Factory function that creates instances of the FrontLED object.

Returns

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

Example Usage:

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

led_example(pFrontLED);

pFrontLED->Release();
```

7.17 IncludeFiles/Lightsensor.h File Reference

Data Structures

- struct Lightsensor

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Lightsensor * LIGHTSENSORHANDLE

Functions

- EXTERN_C CCAUXDLL_API LIGHTSENSORHANDLE CCAUXDLL_CALLING_CONV GetLightsensor (void)

7.17.1 Typedef Documentation

7.17.1.1 typedef CrossControl::Lightsensor* LIGHTSENSORHANDLE

7.17.2 Function Documentation

7.17.2.1 EXTERN_C CCAUXDLL_API LIGHTSENSORHANDLE CCAUXDLL_CALLING_CONV GetLightsensor (void)

Factory function that creates instances of the Lightsensor object.

Returns

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

Example Usage:

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

ls_example(pLightSensor);

pLightSensor->Release();
```

7.18 IncludeFiles/Power.h File Reference

Data Structures

- struct Power

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Power * POWERHANDLE

Functions

- EXTERN_C CCAUXDLL_API POWERHANDLE CCAUXDLL_CALLING_CONV GetPower (void)

7.18.1 Typedef Documentation

7.18.1.1 typedef CrossControl::Power* POWERHANDLE

7.18.2 Function Documentation

7.18.2.1 EXTERN_C CCAUXDLL_API POWERHANDLE CCAUXDLL_CALLING_CONV GetPower (void)

Factory function that creates instances of the Power object.

Returns

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

Example Usage:

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

power_example(pPower);

pPower->Release();
```

7.19 IncludeFiles/PowerMgr.h File Reference

Data Structures

- struct PowerMgr

Namespaces

- namespace CrossControl

Typedefs

- typedef enum CrossControl::PowerMgrConf _PowerMgrConf
- typedef enum CrossControl::PowerMgrStatus _PowerMgrStatus
- typedef CrossControl::PowerMgr * POWERMGRHANDLE

Enumerations

- enum PowerMgrConf { Normal = 0, ApplicationControlled = 1, BatterySuspend = 2 }

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

Functions

- EXTERN_C CCAUXDLL_API
  POWERMGRHANDLE
  CCAUXDLL_CALLING_CONV GetPowerMgr (void)

7.19.1 Typedef Documentation

7.19.1.1 typedef CrossControl::PowerMgr* POWERMGRHANDLE

7.19.2 Function Documentation

7.19.2.1 EXTERN_C CCAUXDLL_API POWERMGRHANDLE CCAUXDLL_CALLING_CONV GetPowerMgr (void)

Factory function that creates instances of the PowerMgr object.

Returns

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

Example Usage:

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

assert(pPowerMgr);
assert(pBattery);

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

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

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

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

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

test_powermgr(pPowerMgr);

pPowerMgr->Release();
```

7.20 IncludeFiles/Releasenotes.dox File Reference

7.21 IncludeFiles/Smart.h File Reference

Data Structures

- struct Smart

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Smart * SMARTHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  SMARTHANDLE
  CCAUXDLL_CALLING_CONV GetSmart (void)

7.21.1 Typedef Documentation

7.21.1.1 typedef CrossControl::Smart* SMARTHANDLE

7.21.2 Function Documentation

7.21.2.1 EXTERN_C CCAUXDLL_API SMARTHANDLE CCAUXDLL_CALLING_CONV GetSmart (void)

Factory function that creates instances of the Smart object.

Returns

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

Example Usage:

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

show_card_data(pSmart);

pSmart->Release();
```

7.22 IncludeFiles/Telematics.h File Reference

Data Structures

- struct Telematics

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Telematics * TELEMATICSHANDLE

Functions

- EXTERN_C CCAUXDLL_API TELEMATICSHANDLE CCAUXDLL_CALLING_CONV GetTelematics (void)

7.22.1 Typedef Documentation

7.22.1.1 typedef CrossControl::Telematics* TELEMATICSHANDLE

7.22.2 Function Documentation

7.22.2.1 EXTERN_C CCAUXDLL_API TELEMATICSHANDLE CCAUXDLL_CALLING_CONV GetTelematics (void)

Factory function that creates instances of the Telematics object.

Returns

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

Example Usage:

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

telematics_example(pTelematics);

pTelematics->Release();
```

7.23 IncludeFiles/TouchScreen.h File Reference

Data Structures

- struct TouchScreen

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::TouchScreen * TOUCHSCREENHANDLE

Enumerations

- enum TouchScreenModeSettings { MOUSE_NEXT_BOOT = 0, TOUCH_NEXT_BOOT = 1, MOUSE_NOW = 2, TOUCH_NOW = 3 }
- enum TSAdvancedSettingsParameter {
  TS_RIGHT_CLICK_TIME = 0, TS_LOW_LEVEL = 1, TS_UNTOUCHLEVEL = 2, TS_DEBOUNCE_TIME = 3,
  TS_DEBOUNCE_TIMEOUT_TIME = 4, TS_DOUBLECLICK_MAX_CLICK_TIME = 5, TS_DOUBLE_CLICK_TIME = 6, TS_MAX_RIGHTCLICK_DISTANCE = 7,
  TS_USE_DEJITTER = 8, TS_CALIBTATION_WIDTH = 9, TS_CALIBRATION_MEASUREMENTS = 10, TS_RESTORE_DEFAULT_SETTINGS = 11
  }

Functions

- EXTERN_C CCAUXDLL_API
  TOUCHSCREENHANDLE
  CCAUXDLL_CALLING_CONV GetTouchScreen (void)

7.23.1 Typedef Documentation

7.23.1.1 typedef CrossControl::TouchScreen* TOUCHSCREENHANDLE

7.23.2 Function Documentation

7.23.2.1 EXTERN_C CCAUXDLL_API TOUCHSCREENHANDLE CCAUXDLL_CALLING_CONV GetTouchScreen (void)

Factory function that creates instances of the TouchScreen object.

Returns

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

Example Usage:

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

touchscreen_example(pTouchScreen);

pTouchScreen->Release();
```

7.24 IncludeFiles/TouchScreenCalib.h File Reference

Data Structures

- struct TouchScreenCalib

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::TouchScreenCalib * TOUCHSCREENCALIBHANDLE

Enumerations

- enum CalibrationModeSettings { MODE_UNKNOWN = 0, MODE_NORMAL = 1, MODE_CALIBRATION_5P = 2, MODE_CALIBRATION_9P = 3, MODE_CALIBRATION_13P = 4 }

- enum CalibrationConfigParam {
  CONFIG_CALIBRATION_WITH = 0, CONFIG_CALIBRATION_MEASUREMENTS = 1, CONFIG_5P_CALIBRATION_POINT_BORDER = 2, CONFIG_13P_CALIBRATION_POINT_BORDER = 3,
  CONFIG_13P_CALIBRATION_TRANSITION_MIN = 4, CONFIG_13P_CALIBRATION_TRANSITION_MAX = 5 }

Functions

- EXTERN_C CCAUXDLL_API
  TOUCHSCREENCALIBHANDLE
  CCAUXDLL_CALLING_CONV GetTouchScreenCalib (void)

7.24.1 Typedef Documentation

7.24.1.1 typedef CrossControl::TouchScreenCalib* TOUCHSCREENCALIBHANDLE

7.24.2 Function Documentation

7.24.2.1 EXTERN_C CCAUXDLL_API TOUCHSCREENCALIBHANDLE CCAUXDLL_CALLING_CONV GetTouchScreenCalib (void)

Factory function that creates instances of the TouchScreenCalib object.

Returns

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

7.25 IncludeFiles/Video.h File Reference

Data Structures

- struct Video

Namespaces

- namespace CrossControl

Typedefs

- typedef CrossControl::Video * VIDEOHANDLE

Functions

- EXTERN_C CCAUXDLL_API
  VIDEOHANDLE
  CCAUXDLL_CALLING_CONV GetVideo (void)

7.25.1 Typedef Documentation

7.25.1.1 typedef CrossControl::Video* VIDEOHANDLE

7.25.2 Function Documentation

7.25.2.1 EXTERN_C CCAUXDLL_API VIDEOHANDLE CCAUXDLL_CALLING_CONV GetVideo (void)

Factory function that creates instances of the Video object.

Returns

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

Index