Texas Instruments bqToolsSDK Communications Manager API

0.0.1.84

Generated by Doxygen 1.7.4

Fri Feb 7 2020 08:22:41

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 COMCHANNELINFO Struct Reference

```
#include <CMAPI.h>
```

Data Fields

- char pipeName [BUF256]
- char dataPipeName [BUF256]
- char priorityPipeName [BUF256]
- char deviceName [BUF256]
- int priorityStatus

3.1.1 Detailed Description

Definition at line 38 of file CMAPI.h.

3.1.2 Field Documentation

3.1.2.1 char **COMCHANNELINFO::pipeName**[BUF256]

Definition at line 40 of file CMAPI.h.

3.1.2.2 char **COMCHANNELINFO::dataPipeName**[BUF256]

Definition at line 41 of file CMAPI.h.

3.1.2.3 char COMCHANNELINFO::priorityPipeName[BUF256]

Definition at line 42 of file CMAPI.h.

3.1.2.4 char COMCHANNELINFO::deviceName[BUF256]

Definition at line 43 of file CMAPI.h.

3.1.2.5 int COMCHANNELINFO::priorityStatus

Definition at line 44 of file CMAPI.h.

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

- CMAPI.h

Chapter 4

File Documentation

4.1 CMAPI.h File Reference

Data Structures

- struct COMCHANNELINFO

Defines

- #define BUFSIZE 2000
- #define BUF256 256
- #define FOREGROUND 0
- #define BACKGROUND 1
- #define SMBUS 0
- #define I2C 1
- #define HDQ 2
- #define SDQ 3
- #define SMB_I2C 4
- #define I2CCRC 5
- #define SPI 6
- #define SPICRC 7
- #define EXPTYPE __declspec(dllimport)

Functions

- EXPTYPE int SDKReadSMBWord (const char ∗dataPipeName, short command, short ∗wordRead, short targetAddress)

  Write a word over SMB.
- EXPTYPE int SDKReadSMBBlock (const char ∗dataPipeName, short command, unsigned char ∗dataRead, short ∗numBytesRead, int bufferSize, short targetAddress)

Read a block of data over SMB.

- EXPTYPE int SDKWriteSMBWord (const char ∗dataPipeName, short command, short wordtoWrite, short delayAfterWord, short targetAddress)

  Write a word to SMB.
- EXPTYPE int SDKWriteSMBCommand (const char ∗dataPipeName, short command, short delayAfterCommand, short targetAddress)

  Write an SMB command.
- EXPTYPE int SDKWriteSMBBlock (const char ∗dataPipeName, short command, unsigned char ∗datatoWrite, short numBytestoWrite, short delayAfterBlock, short targetAddress)

  Write a block of data to SMB.
- EXPTYPE int SDKWriteSMBWordReadBlock (const char ∗dataPipeName, short wordCommand, int wordtoWrite, short delayAfterWord, short blockCommand, unsigned char ∗dataRead, int bufferSize, short ∗numBytesRead, short targetAddress)

  Write a command with specified delay in msec after sending command. Then read SMB block. In most cases, the delay will be zero.
- EXPTYPE int SDKWriteSMBWordWriteBlock (const char ∗dataPipeName, short wordCommand, int wordtoWrite, short delayAfterWord, short blockCommand, unsigned char ∗bytestoWrite, int numBytestoWrite, short targetAddress)

  Write a MAC command with specified delay in msec after sending command. Then write SMB block.
- EXPTYPE int SDKWriteSMBBlkReadBlk (const char ∗dataPipeName, short firstBlkStartAddress, unsigned char ∗firstBlkData, short firstNumBytestoWrite, short delayAfterFirstBlock, short block2StartAddress, unsigned char ∗dataRead, int bufferSize, short ∗numBytesRead, short targetAddress)

  Write first block with number of bytes specified with delay in msec after sending first block. Then read a block of of data of specified length. Maximum length of first block is 6. Length of second block must be less than 96 bytes for EV2300.
- EXPTYPE int SDKI2CReadBlock (const char ∗dataPipeName, short startAddress, unsigned char ∗dataRead, short numofBytestoRead, int bufferSize, short targetAddress)

  I2C Read Block Start address can be skipped optionally and needs to be handled. Target address can be skipped. If bit 15 of target addess is set, target address will not be sent.
- EXPTYPE int SDKI2CWriteBlock (const char ∗dataPipeName, short startAddress, unsigned char ∗dataToWrite, short numBytestoWrite, short delayAfterWrite, short targetAddress)

  I2C Write a block for data over I2C. 7 bit target address is default. If bit 15 of startAddress is high, the target address is interperted as a 10 bit address. If bit 15 of target addess is set target address will not be sent. This is not implemented yet in the packet layer.
- EXPTYPE int SDKI2CLtReadBlock (const char ∗dataPipeName, short nWordAddr, short setTestMode, unsigned char ∗DataBlock, short numBytestoRead, int bufferSize)

  Read an I2C Light block of bytes over I2C. EV2400 communications adapter only.

- EXPTYPE int SDKI2CLtWriteBlock (const char ∗dataPipeName, short nWordAddr, short setTestMode, unsigned char ∗DataBlock, short nLen, short delayAfterWrite)

  Write an I2C Light block over I2C. Maximum block size is 32 bytes. EV2400 communication adapter only.
- EXPTYPE int SDKI2CLtStop (const char ∗dataPipeName)

  Send an I2C Light stop. EV2400 communication adapter only.
- EXPTYPE int SDKI2CLtSetTiming (const char ∗dataPipeName, unsigned char tHigh, unsigned char tLow, unsigned char tSep, unsigned char tIdle, unsigned short tSynch)

  Set I2C Light timing characteristics. EV2400 adapter only.
- EXPTYPE int SDKI2CCRCReadBlock (const char ∗dataPipeName, short startAddress, unsigned char ∗dataRead, short numofBytestoRead, int bufferSize, short targetAddress)

  Read a block of data over I2C with bq769x2 CRC codes. bq769x2 sends a CRC verification byte with every data byte. This function will strip the CRC codes and return the only the data in dataRead.
- EXPTYPE int SDKI2CCRCWriteBlock (const char ∗dataPipeName, short startAddress, unsigned char ∗dataToWrite, short numBytestoWrite, short delayAfterWrite, short targetAddress)

  Write a block of data over I2C using bq769x2 CRC mode. A CRC is sent after every byte in this mode. See bq769x2 datasheet for details. This function adds the CRC bytes based the dataToWrite and length. Only pass the data without CRC to this function.
- EXPTYPE int SDKHDQ8ReadBlock (const char ∗dataPipeName, short startAddress, unsigned char ∗dataRead, short numBytestoRead, int bufferSize)

  Write a block of data using HDQ8 protocol.
- EXPTYPE int SDKHDQ8WriteBlock (const char ∗dataPipeName, short startAddress, unsigned char ∗dataToWrite, short numBytestoWrite)

  Read a block of data using HDQ8 protocol.
- EXPTYPE int SDKI2CWriteBlkReadBlk (const char ∗dataPipeName, short firstBlkStartAddress, unsigned char ∗firstBlkData, short firstNumBytestoWrite, short delayAfterFirstBlock, short block2StartAddress, unsigned char ∗dataRead, int bufferSize, short numBytestoRead, short targetAddress)

  Write first block with number of bytes specified with delay in msec after sending first block. Then read a block of of data of specified length. Length must be less than 96 bytes for EV2300. In most cases, the delay between commands will be zero.
- EXPTYPE int SDKI2CWriteBlkWriteBlk (const char ∗dataPipeName, short firstBlkStartAddress, unsigned char ∗firstBlkData, short firstNumBytestoWrite, short delayAfterFirstBlock, short block2StartAddress, unsigned char ∗datatoWrite, short numBytestoWrite, short delayAfterWriteBlock, short targetAddress)

  Write a block then Write a block over I2C. This will return immediately if the first block write fails. In most cases, the delay between commands will be zero.
- EXPTYPE int SDKReadWriteSubClsBlock (const char ∗dataPipeName, short subClassNumber, short blockNumber, unsigned char ∗dataBlock, short numBytes, short operation, short verifyOperation, short delayAfterCmd, short protocol, short targetAddress)

This will read or write one bank of single cell data flash using SubClass data flash access. This function will read or write based on the flag "operation". Operation = 0 is a read and operation = 1 is a write. The supported protocols are I2C and HDQ only. A bank of data flash is 32 bytes in length. Operations begin at the beginning of the specified bank. A delay in milliseconds can be specified after the block operation. A built in delay of 180 milliseconds in this function after a write operation to allow time for the data to be written to flash. The delay parameter will be in addtion to this delay. The write operation will be verified if verifyOperation = 1. Read operations are not verified but will report and error if communications fail. Parameters not relevant to HDQ will be ignored. Write operations. All 32 bytes must be written to a block. If the subclass contains less then 32 bytes, the unused bytes in the block must be set to zero. The dataBlock must contain all 32 bytes. numBytes must be 32 for a write operation. Write verification reads the data block data back and compares to what was written.

- EXPTYPE int SDKWriteMAC2Words (const char ∗dataPipeName, short firstStartAddress, unsigned short firstWordtoWrite, short delayAfterFirstCommand, short secondStartAddress, unsigned short secondWordtoWrite, short delayAfterSecondCommand, short MACStyle, short protocol, short targetAddress)

  This function will write two words. The primary use of this function is to write two unseal words. The MACStyle parameter will send the commands in old MAC style if MACStyle = 0. Words are sent new MAC style if MACStyle = 1 or any other value. The supported protocols are SMB = 0 I2C = 1, and HDQ = 2. Targetaddress is not valid for HDQ and will be ignored.

- EXPTYPE int SDKGetUSBAdapterVersion (const char ∗dataPipeName, char ∗version)

  Read EV2X00 Firmware version. This may be used to test communication between host and adapter EV2300 and EV2400 use a different format for version. Version is returned char ∗. The buffer passed must be allocated with a length of greater than 10 bytes.

- EXPTYPE void SDKCloseDevice (const char ∗dataPipeName)

  Close Adapter Handle This will close adapter handle. Communication with the device continues but adapter is free to be reopened. The current adapter reported by CM will be a null string after this command is sent.

- EXPTYPE int DiscoverNamesofCM (char ∗cmexecuting, int ∗totalCm)

  Discover names and total number of CM processes currently executing on system. Command channel pipe server names are returned in a comma separated CString Recommended buffer length is 2000 bytes.

- EXPTYPE int DiscoverAllCMProcesses (int ∗totalCm)

  Returns number of running CM processes executing on system. This is done by probing each command channel.

- EXPTYPE int CreateChildProcess (char ∗cmdline, int startupMode)

  Start a new stand alone process with CreateProcess. Pass entire command line including arguments without -c option. Example command line is "c:\\tmp\\commmgr.exe" or "c:\\tmp\\commmgr.exe -l c:\\tmp\\logfile.txt" Process will start either in a console in the background.

- EXPTYPE int DoesCMUsingCmdChannelExist (const char ∗commandChannelName)

  Test if an instance of commmgr.exe is executing using command channel pipe name.

- EXPTYPE char ∗ TranslateCommError (int errorCode)

  Translate a BMS tools communication error code This translates EV2X00 adapter, updater, and tools SDK error codes to text.

- EXPTYPE void RegisterPIDwithCM (const char ∗cmdPipeName)

  Register a process by sending process ID to CM. The CM keeps a record of all process IDs used by connection processes to determine when CM can terminate. A new process must register once with each instance of a CM that it will use. The CM will terminate when all registered process IDs no longer exist on system.
- EXPTYPE int TerminateCM (const char ∗cmdPipeName)

  Terminate CM process using this command channel pipe name.
- EXPTYPE int SetupAppwithFirstFreeAdapter (char ∗cmdPipeName, char ∗dataPipeName, char ∗priorityPipeName, char ∗evDeviceName, char ∗exePathwithArgs, int buffersize, int processCreateStyle)

  Spawn a new commmgr.exe on an unused command channel. Open first free adapter. Prepare for EVM communication. Returns all pipe names and EV2X00 device name opened. CM process will spawn in background or foreground based on value of processCreateStyle. This is equivalant to how current tools obtain any free adapter handle This function will kill the created CM if no free adapters are available or any other error.
- EXPTYPE int UseSpecificEV2X00 (const char ∗cmdPipeName, const char ∗evDeviceName)

  Use specific EV2X00 device with a communication manager. This will close current device and use the device file name specified in parameter if it is free. An EV2400 adapter always uses the same HID device name. An EV2300 adapter always uses a different name.
- EXPTYPE int IsCmdChannelNameValid (const char ∗cmdPipeName)

  Verify that command channel pipe name is a well known name.
- EXPTYPE int GetDataandPriorityNames (const char ∗cmdPipeName, char ∗dataPipeName, char ∗priorityPipeName, int bufferSize)

  Get data and priority pipe names for an executing communication manager process using this command channel name.
- EXPTYPE int GetCMAdapterHandle (const char ∗cmdPipeName, char ∗evDeviceName)

  Returns the name of adapter handle device file name used by an executing communication manager on command pipe name. If CM is executing but does not have a handle, parameter evDeviceName returned will be a null string. Use a minimum buffer length of 256 bytes.
- EXPTYPE int GetPriorityChannelState (const char ∗cmdPipeName, int ∗status)

  This function queries and returns the state of the priority channel of an executing CM. Valid state are OFF=0, ON=1, CANCELLED=2.
- EXPTYPE int StartPriorityChanSequence (const char ∗cmdPipeName)

  Begin a priority channel sequence of EVM commands. This will return an error if a priority sequence is already in progress. The data channel will be suspended until an EndPriorityChanSequence is sent to release this channel.
- EXPTYPE int EndPriorityChanSequence (const char ∗cmdPipeName)

  End a priority channel sequence of EVM commands. This will terminate the priorty channel and release pending data channel operations in progress. Data channel requests will be serviced. CM returns to normal operation.
- EXPTYPE int SetPriorityChanKeepAlive (const char ∗cmdPipeName)

  Set priority keep alive. This keep alive is sent at a minimum of once a second for a long proiroty channel sequence. The priority channel will automatically cancel a priority sequence and return to normal operation if not refreshed with a keepalive packet. An *example of is a long sequence of commands flashing a device. This command is required to be sent at least once per second to keep priority channel alive.*

- EXPTYPE int GetSystemInventory (COMCHANNELINFO cmStats[ ])

  This scans the entire system and reports executing CM processes, channel names, adapter handles, and adapter device name for each unique command channel. All well known command channel names are probed. The result is return in an array of COMCHANNELINFO structures. The address of the structure defined as COMCHANNELINFO sysInv[MAXCMPROCESSES] is passed as the first parameter. Structure will be updated based on max number of CMs. Data, priority pipe name, and adapter device name will be null for a command pipe that is unassigned and available.
- EXPTYPE int GetAllFreeAdapters (const char ∗cmdPipeName, char ∗adapterBuffer)

  Get all free EV2X00 adapters connected and ready to use on a system. These are searched using an already executing CM. This returns a comma separated list of adapter free adapter device names. Buffer length to hold adapter names should be a minimum of 1900 bytes. This will report the first 16 free adapters found.
- EXPTYPE int ResetCM (const char ∗cmdPipeName)

  Reset communication manager to initial state. This function will cancel a priority sequence if in progress and close adapter handle.
- EXPTYPE int GetCMReleaseVersion (const char ∗cmdPipeName, char ∗versionString)

  Returns the release version of the communication manager in a version string. CString returned will not be greater than 40 bytes in length.
- EXPTYPE int DoesCMHaveHandle (const char ∗cmdPipeName)

  Does CM have an adapter handle.
- EXPTYPE int SDKSetPinVoltage (const char ∗dataPipeName, short virtualPinNumber, short voltageLevelmV)
- EXPTYPE int SDKGetBoardType (const char ∗dataPipeName, unsigned char ∗boardType, unsigned short ∗numBytesRead)
- EXPTYPE int SDKSetVoutWithTimeout (const char ∗dataPipeName, unsigned char vout1Enable, unsigned char vout2Enable, unsigned char vout3Enable, unsigned char vout4Enable)
- EXPTYPE int SDKSetPWMConfig (const char ∗dataPipeName, unsigned short dutyCycle)

  Set PWM Configuration.
- int SendRecCommand (const char ∗dataPipeName, int cmd, char ∗messagetoSend, char ∗messageRecv)

  Send and receive a command packet to a command pipe server. Each command packet has a unique format. The command packet format description is in the Communication Manager design document. This is primarily for testing. Message format as defined in COMMESSAGE type Both sent message and returned message will be in COMMESSAGE format A buffer size of 1900 bytes is recommended. This is used for debugging.
- EXPTYPE int IsDataPipeNameValid (const char ∗dataPipeName)
- EXPTYPE int C816Senc (const char ∗dataPipeName, const char ∗srecFileName, int operation, int protocol)

  The will program or create a C816 senc from connected device. These are generally single cell devices like bq27541. ROM version is not checked. Device must be executing ROM prior to call. SMB supported family is bq8030.

- EXPTYPE int ProgramC816Senc_I2C (const char ∗dataPipeName, const char ∗srecFileName)
- EXPTYPE int SDKGetEV2300Name (const char ∗dataPipeName, char ∗ev2300Name)

  Get name stored in an EV2300 communications adapter. This will return a previously stored name in EV2300 data flash that is used as an identifier. The expected length of this name will be 32 bytes or less. A pointer to a pre-allocated byte buffer of at least 33 bytes in length is passed. A CString with the EV2300 name will reside in this buffer after the call. Use the function SDKSetEV2300Name to write a name into an EV2300. This function does nothing if adapter is an EV2400. Each EV2400 has a unique serial number as shipped that can be used a unique identifier. This serial number is also available in the Windows Device Manager under Human Interface Devices HID-compliant device for the adapter. The "Parent" property contains this serial number. EV2400 version 0.27 and later firmware allows setting a unique name also in EV2400 flash. If a name is set, it overrides the serial number. Bq80xrw.dll version 0.1.69 and later allows EV2400 serial number to be returned in a hexadecimal string with the function SDKGetEV2300Name if name has been set using another tool. An example of an EV2400 serial number is 7624996F27002A00. The Parent propety in the device manager for this device is USB\VID_0451&PID_0037\7624996F27002A00.
- EXPTYPE int SDKSetEV2300Name (const char ∗dataPipeName, char ∗ev2300Name)

  Set EV2300 name writes a string to data flash inside and EV2300 communication adapter. This is used as an identifier. This name will be truncated if larger than 31 bytes. Names cannot be stored in an EV2400 adapter. Each EV2400 has a unique serial number based on die ID that will not change. This is the EV2400 identifier.
- EXPTYPE int SDKSetAdapterI2CClockSpeed (const char ∗dataPipeName, int speed)

  Set EV2300 or EV2400 I2C clock speed.
- EXPTYPE void setBMDKConstants (short extPort, int isExtPort, short intPort, int isIntPort, int ocHandle, int isHandle)

  Sets BMDK Configuration parameters.
- EXPTYPE int SDKSetEV2400CommClkSpeed (const char ∗dataPipeName, short clkSpeedkHz, short tgtProtocol)

  Set EV2400 protocol clock speed for SMB or I2C in kHz.
- EXPTYPE int SDKSPIConfigure (const char ∗dataPipeName, short polarity, short phase, int clkSpeedkHz, short SPICRCEnabled)

  Configure SPI EV2400 master characteristics. EV2400 with firmware version 0.30 or later is required.
- EXPTYPE int SDKSPIReadConfiguration (const char ∗dataPipeName, short ∗polarity, short ∗phase, int ∗clkSpeedkHz, short ∗SPICRCEnabled)

  Read EV2400 SPI master characteristics. EV2400 with firmware version 0.30 or later is required.
- EXPTYPE int SDKSPI16ReadBlock (const char ∗dataPipeName, short startAddress, unsigned char ∗dataRead, short numBytestoRead)

  Read a SPI16 Block. EV2400 with firmware version 0.30 or later is required.
- EXPTYPE int SDKSPI16WriteBlock (const char ∗dataPipeName, short startAddress, unsigned char ∗datatoWrite, short numBytestoWrite, short delayAfterWrite)

  Write a SPI16 Block. EV2400 with firmware version 0.30 or later is required.

- EXPTYPE int SDKSPI16WriteBlkReadBlk (const char ∗dataPipeName, short firstBlkStartAddress, unsigned char ∗firstBlkData, short firstNumBytestoWrite, short delayAfterFirstBlock, short block2StartAddress,\unsigned char ∗dataRead, short numBytestoRead)

  This is used to send a MAC subcommmand and read block of data return by subcommand. SPI Write first block with number of bytes specified with delay in msec after sending first block. Then read a block of of data of specified length. Length must be less than 128 bytes for EV2400. Write data can be up to 4 bytes but only 2 are required for a MAC subcommand. Delay after write is used to give MAC command time to execute. These commands are atomic. Both will be executed and cannot be interrupted by another thread.
- EXPTYPE int SPIAA_Control (const char ∗dataPipeName, int operation, unsigned int serialNumber)

  Internal function to control Aardvark SPI for debugging.
- EXPTYPE int SDKUseAardvarkSPI (const char ∗dataPipeName, int flag)

4.1.1 Define Documentation

4.1.1.1 #define BUFSIZE 2000

Common buffer size

Definition at line 17 of file CMAPI.h.

4.1.1.2 #define BUF256 256

Definition at line 18 of file CMAPI.h.

4.1.1.3 #define FOREGROUND 0

Spawn process behavior for functions. SetupAppwithFirstFreeAdapter and CreateChild-Process

Definition at line 24 of file CMAPI.h.

4.1.1.4 #define BACKGROUND 1

Definition at line 25 of file CMAPI.h.

4.1.1.5 #define SMBUS 0

Definition at line 28 of file CMAPI.h.

4.1.1.6 #define I2C 1

Definition at line 29 of file CMAPI.h.

4.1.1.7 #define HDQ 2

Definition at line 30 of file CMAPI.h.

4.1.1.8 #define SDQ 3

Definition at line 31 of file CMAPI.h.

4.1.1.9 #define SMB_I2C 4

Definition at line 32 of file CMAPI.h.

4.1.1.10 #define I2CCRC 5

Definition at line 33 of file CMAPI.h.

4.1.1.11 #define SPI 6

Definition at line 34 of file CMAPI.h.

4.1.1.12 #define SPICRC 7

Definition at line 35 of file CMAPI.h.

4.1.1.13 #define EXPTYPE __declspec(dllimport)

Definition at line 56 of file CMAPI.h.

4.1.2 Function Documentation

4.1.2.1 EXPTYPE int SDKReadSMBWord (const char ∗ *dataPipeName,* short *command,* short ∗ *wordRead,* short *targetAddress*)

Write a word over SMB.

Parameters

dataPipeName	Name of data pipe
command	SMB command
wordRead	SMB word read
targetAddress	SMB slave address

Returns

Non zero if error or warning

4.1.2.2 EXPTYPE int SDKReadSMBBlock (const char ∗ *dataPipeName,* short *command,* unsigned char ∗ *dataRead,* short ∗ *numBytesRead,* int *bufferSize,* short *targetAddress*)

Read a block of data over SMB.

Parameters

dataPipeName	Name of data pipe
command	SMB command
dataRead	Buffer to hold bytes to write
numBytesRead	Number of bytes actually read
bufferSize	Size of buffer
targetAddress	SMB slave address

Returns

Non zero if error or warning

4.1.2.3 EXPTYPE int SDKWriteSMBWord (const char ∗ *dataPipeName,* short *command,* short *wordtoWrite,* short *delayAfterWord,* short *targetAddress*)

Write a word to SMB.

Parameters

dataPipeName	Name of data pipe
command	SMB command
wordtoWrite	word to write
delayAfterWord	Delay in msec after word is sent
targetAddress	short SMB slave address

Returns

Non zero if error or warning

4.1.2.4 EXPTYPE int SDKWriteSMBCommand (const char ∗ *dataPipeName,* short *command,* short *delayAfterCommand,* short *targetAddress*)

Write an SMB command.

Parameters

dataPipeName	Name of data pipe
command	SMB command
delayAfterCommand	Delay in msec after commmand is sent
targetAddress	SMB slave address

Returns

Non zero if error or warning

4.1.2.5 EXPTYPE int SDKWriteSMBBlock (const char ∗ *dataPipeName,* short *command,* unsigned char ∗ *datatoWrite,* short *numBytestoWrite,* short *delayAfterBlock,* short *targetAddress*)

Write a block of data to SMB.

Parameters

dataPipeName	Name of data pipe
command	SMB command
datatoWrite	buffer holding bytes to write
numBytestoWrite	Number of bytes to write.
delayAfterBlock	Delay in msec after sending block
targetAddress	SMB slave address

Returns

Non zero if error or warning

4.1.2.6 EXPTYPE int SDKWriteSMBWordReadBlock (const char ∗ *dataPipeName,* short *wordCommand,* int *wordtoWrite,* short *delayAfterWord,* short *blockCommand,* unsigned char ∗ *dataRead,* int *bufferSize,* short ∗ *numBytesRead,* short *targetAddress*)

Write a command with specified delay in msec after sending command. Then read SMB block. In most cases, the delay will be zero.

Parameters

dataPipeName	Name of data pipe
wordCommand	Word command
wordtoWrite	word to write
delayAfterWord	Delay after sending word command before executing Read Block Command
blockCommand	Command of block to read.
dataRead	buffer to store returned bytes read from device
bufferSize	size of buffer that passed to hold returned bytes
numBytesRead	Number of bytes read
targetAddress	SMB slave address

Returns

Non zero if error or warning from read block command

4.1.2.7 EXPTYPE int SDKWriteSMBWordWriteBlock (const char ∗ *dataPipeName,* short *wordCommand,* int *wordtoWrite,* short *delayAfterWord,* short *blockCommand,* unsigned char ∗ *bytestoWrite,* int *numBytestoWrite,* short *targetAddress*)

Write a MAC command with specified delay in msec after sending command. Then write SMB block.

Parameters

dataPipeName	Name of data pipe
wordCommand	MAC command
wordtoWrite	word to write
delayAfterWord	Delay after sending word in msec
blockCommand	Command to write
bytestoWrite	Buffer holding bytes to write
numBytestoWrite	Number of bytes to write
delayAfterWord	Delay in msec after write block command
targetAddress	SMB slave address

Returns

Non zero if error or warning from write block command

4.1.2.8 EXPTYPE int SDKWriteSMBBlkReadBlk (const char ∗ *dataPipeName,* short *firstBlkStartAddress,* unsigned char ∗ *firstBlkData,* short *firstNumBytestoWrite,* short *delayAfterFirstBlock,* short *block2StartAddress,* unsigned char ∗ *dataRead,* int *bufferSize,* short ∗ *numBytesRead,* short *targetAddress*)

Write first block with number of bytes specified with delay in msec after sending first block. Then read a block of of data of specified length. Maximum length of first block is 6. Length of second block must be less than 96 bytes for EV2300.

Parameters

dat-aPipeName	Name of data pipe
firstBlkStar-tAddress	Start address of first block to to write.
firstBlkData	First block of data. Data to write to first block.
firstNum-BytestoWrite	Number of bytes to write. This must be a number from 1 to 6.
delayAfter-FirstBlock	Delay in milliseconds after sending first block before beginning to send sec-ond block
block2StartAc	Start Address of block to read.
dataRead	buffer to store returned bytes read from device
bufferSize	size of buffer that is passed to hold returned bytes
∗*numBytesRe*	Number of bytes read from second block
targetAd-dress	SMB slave address

Returns

Non zero if error or warning from read block command

4.1.2.9 EXPTYPE int SDKI2CReadBlock (const char ∗ *dataPipeName,* short *startAddress,* unsigned char ∗ *dataRead,* short *numofBytestoRead,* int *bufferSize,* short *targetAddress*)

I2C Read Block Start address can be skipped optionally and needs to be handled. Target address can be skipped. If bit 15 of target addess is set, target address will not be sent.

Parameters

dat-aPipeName	Name of data pipe

startAddress	Start address of data to read
dataRead	Buffer to hold returned data
numofByte-stoRead	Number of bytes to read
bufferSize	of buffer to hold returned data
targetAd-dress	I2C slave address

Returns

Non zero if error or warning

4.1.2.10 EXPTYPE int SDKI2CWriteBlock (const char ∗ *dataPipeName,* short *startAddress,* unsigned char ∗ *dataToWrite,* short *numBytestoWrite,* short *delayAfterWrite,* short *targetAddress*)

I2C Write a block for data over I2C. 7 bit target address is default. If bit 15 of startAddress is high, the target address is interperted as a 10 bit address. If bit 15 of target addess is set target address will not be sent. This is not implemented yet in the packet layer.

Parameters

dat-aPipeName	Name of data pipe
startAddress	Start address to begin write
dataToWrite	Bytes to write
num-BytestoWrite	Number of bytes to write
delayAfter-Write	delay in msec after write
targetAd-dress	I2C slave address

Returns

Non zero if error or warning

4.1.2.11 EXPTYPE int SDKI2CLtReadBlock (const char ∗ *dataPipeName,* short *nWordAddr,* short *setTestMode,* unsigned char ∗ *DataBlock,* short *numBytestoRead,* int *bufferSize*)

Read an I2C Light block of bytes over I2C. EV2400 communications adapter only.

Parameters

dat-aPipeName	Name of data pipe

nWordAddr	Address
setTestMode	Set test mode if not equal to 0.
DataBlock	buffer to store result of read.
numBytestoRead	Number of bytes to read from device.
bufferSize	Size of buffer to hold returned data.

Returns

Zero for success. Non zero for error.

4.1.2.12 EXPTYPE int SDKI2CLtWriteBlock (const char ∗ *dataPipeName,* short *nWordAddr,* short *setTestMode,* unsigned char ∗ *DataBlock,* short *nLen,* short *delayAfterWrite*)

Write an I2C Light block over I2C. Maximum block size is 32 bytes. EV2400 communication adapter only.

Parameters

dataPipeName	Name of data pipe
nWordAddr	Address
setTestMode	Set test mode if not equal to 0.
DataBlock	buffer of actual bytes to write. 32 bytes or less.
nLen	number of bytes to write.
delayAfterWrite	Delay in msec after write.

Returns

Zero for success

4.1.2.13 EXPTYPE int SDKI2CLtStop (const char ∗ *dataPipeName*)

Send an I2C Light stop. EV2400 communication adapter only.

Parameters

dataPipeName	Name of data pipe

Returns

Zero for success

4.1.2.14 EXPTYPE int SDKI2CLtSetTiming (const char ∗ *dataPipeName,* unsigned char *tHigh,* unsigned char *tLow,* unsigned char *tSep,* unsigned char *tIdle,* unsigned short *tSynch*)

Set I2C Light timing characteristics. EV2400 adapter only.

Parameters

dataPipeName	Name of data pipe
tHigh	High time of I2C SCL line in microsenonds.
tLow	Low time of I2C SCL line in microseconds.
tSep	Minimum time between SCL and SDA edges in microseconds.
tIdle	Minimum time between a Stop and Stop condition in microseconds.
tSynch	The high time of SCL after the ACK/NACK before a Stop condition in microseconds. This is a 16 bit value.

Returns

Zero for success. Non-zero error or warning

4.1.2.15 EXPTYPE int SDKI2CCRCReadBlock (const char ∗ *dataPipeName,* short *startAddress,* unsigned char ∗ *dataRead,* short *numofBytestoRead,* int *bufferSize,* short *targetAddress*)

Read a block of data over I2C with bq769x2 CRC codes. bq769x2 sends a CRC verification byte with every data byte. This function will strip the CRC codes and return the only the data in dataRead.

Parameters

dataPipeName	Name of data pipe
startAddress	Start address of data to read
dataRead	Buffer to hold returned data
numofBytestoRead	Number of data bytes to read. Do not include CRC bytes.
bufferSize	of buffer to hold returned data
targetAddress	I2C slave address

Returns

Non zero if error or warning

4.1.2.16 EXPTYPE int SDKI2CCRCWriteBlock (const char ∗ *dataPipeName,* short *startAddress,* unsigned char ∗ *dataToWrite,* short *numBytestoWrite,* short *delayAfterWrite,* short *targetAddress*)

Write a block of data over I2C using bq769x2 CRC mode. A CRC is sent after every byte in this mode. See bq769x2 datasheet for details. This function adds the CRC bytes based the dataToWrite and length. Only pass the data without CRC to this function.

Parameters

dataPipeName	Name of data pipe
startAddress	Start address to begin write
dataToWrite	Bytes to write
numBytestoWrite	Number of bytes to write
delayAfterWrite	delay in msec after write
targetAddress	I2C slave address

Returns

Non zero if error or warning

4.1.2.17 EXPTYPE int SDKHDQ8ReadBlock (const char ∗ *dataPipeName,* short *startAddress,* unsigned char ∗ *dataRead,* short *numBytestoRead,* int *bufferSize*)

Write a block of data using HDQ8 protocol.

Parameters

dataPipeName	Name of data pipe to send command
startAddress	Start Address of block read.
dataRead	Buffer to hold returned data
numBytestoRead	Number of Bytes read
bufferSize	Size of dataRead buffer that holds returned data.

Returns

int Non zero for errors or warnings

4.1.2.18 EXPTYPE int SDKHDQ8WriteBlock (const char ∗ *dataPipeName,* short *startAddress,* unsigned char ∗ *dataToWrite,* short *numBytestoWrite*)

Read a block of data using HDQ8 protocol.

Parameters

dataPipeName	Name of data pipe
startAddress	Start address to begin write
dataToWrite	Bytes to write
numBytestoWrite	Number of bytes to write

Returns

Non zero if error or warning

4.1.2.19 EXPTYPE int SDKI2CWriteBlkReadBlk (const char ∗ *dataPipeName,* short *firstBlkStartAddress,* unsigned char ∗ *firstBlkData,* short *firstNumBytestoWrite,* short *delayAfterFirstBlock,* short *block2StartAddress,* unsigned char ∗ *dataRead,* int *bufferSize,* short *numBytestoRead,* short *targetAddress*)

Write first block with number of bytes specified with delay in msec after sending first block. Then read a block of of data of specified length. Length must be less than 96 bytes for EV2300. In most cases, the delay between commands will be zero.

Parameters

dataPipeName	Name of data pipe
firstBlkStartAddress	Start address of first block to to write.
firstBlkData	First block of data. The can be 1, 2, 3, or 4 bytes in length
firstNumBytestoWrite	Number of bytes to write. This must be a number from 1 to 4.
delayAfterFirstBlock	Delay after sending first block before beginning to send second block
block2StartAc	Start Address of block to read.
dataRead	buffer to store returned bytes read from device
bufferSize	size of buffer that is passed to hold returned bytes
numBytestoRead	Number of bytes read from second block
targetAddress	SMB slave address

Returns

Non zero if error or warning from read block command

4.1.2.20 EXPTYPE int SDKI2CWriteBlkWriteBlk (const char ∗ *dataPipeName,* short *firstBlkStartAddress,* unsigned char ∗ *firstBlkData,* short *firstNumBytestoWrite,* short *delayAfterFirstBlock,* short *block2StartAddress,* unsigned char ∗ *datatoWrite,* short *numBytestoWrite,* short *delayAfterWriteBlock,* short *targetAddress*)

Write a block then Write a block over I2C. This will return immediately if the first block write fails. In most cases, the delay between commands will be zero.

Parameters

dat-aPipeName	Name of data pipe
firstBlkStar-tAddress	First block start address.
firstBlkData	First block data.
firstNum-BytestoWrite	Number of first block bytes to write. Valid size is 1-4 bytes. In most cases this will be 2
delayAfter-FirstBlock	Delay after sending first block write before executing next write block write in msec.
block2StartAc	Start Address of 2nd block to write.
datatoWrite	buffer to store returned bytes read from device
num-BytestoWrite	Number of bytes to write
delayAfter-WriteBlock	Delay in milliseconds after writing block before function returns
targetAd-dress	slave address

Returns

Non zero if error or warning from read block command

4.1.2.21 EXPTYPE int SDKReadWriteSubClsBlock (const char ∗ *dataPipeName,* short *subClassNumber,* short *blockNumber,* unsigned char ∗ *dataBlock,* short *numBytes,* short *operation,* short *verifyOperation,* short *delayAfterCmd,* short *protocol,* short *targetAddress*)

This will read or write one bank of single cell data flash using SubClass data flash access. This function will read or write based on the flag "operation". Operation = 0 is a read and operation = 1 is a write. The supported protocols are I2C and HDQ only. A bank of data flash is 32 bytes in length. Operations begin at the beginning of the specified bank. A delay in milliseconds can be specified after the block operation. A built in delay of 180 milliseconds in this function after a write operation to allow time for the data to be written to flash. The delay parameter will be in addtion to this delay. The write operation will be verified if verifyOperation = 1. Read operations are not verified but will report and error if communications fail. Parameters not relevant to HDQ will be ignored. Write operations. All 32 bytes must be written to a block. If the subclass contains less then 32 bytes, the unused bytes in the block must be set to zero. The dataBlock must contain all 32 bytes. numBytes must be 32 for a write operation. Write verification reads the data block data back and compares to what was written.

Parameters

dataPipeName	Name of data pipe to use.
subClassNumber	Number of subclass to access
blockNumber	This is the block number to access. The first block is block 0.
∗*dataBlock*	Array of unsigned bytes that contains the data to write or storage for bytes read. This must be allocated and a minimum of 32 bytes in length.
numBytes	Number of bytes to read or write. This must be 32 for write operations. Read operation will return the number of bytes starting from the beginning of the block.
operation	0 = read SubClass block, 1 = write SubClass Block
verifyOperation	1 = verify operation. Setting to 1 will verify the write operation. Read is not verified.
delayAfterCmd	This is a delay in milliseconds after the block command is sent. A delay of 180 milliseconds is already present after a write operation.
protocol	1 = I2C, 2 = HDQ No other values are valid
targetAddress	I2C address of device. Not used with HDQ

Returns

Non zero if error or warning

4.1.2.22 EXPTYPE int SDKWriteMAC2Words (const char ∗ *dataPipeName,* short *firstStartAddress,* unsigned short *firstWordtoWrite,* short *delayAfterFirstCommand,* short *secondStartAddress,* unsigned short *secondWordtoWrite,* short *delayAfterSecondCommand,* short *MACStyle,* short *protocol,* short *targetAddress*)

This function will write two words. The primary use of this function is to write two unseal words. The MACStyle parameter will send the commands in old MAC style if MACStyle = 0. Words are sent new MAC style if MACStyle = 1 or any other value. The supported protocols are SMB = 0 I2C = 1, and HDQ = 2. Targetaddress is not valid for HDQ and will be ignored.

Parameters

dataPipeName	Name of data pipe to use.
firstStartAddress	Address (MAC) to begin writing first word.
firstWordtoWrite	First word to write. This is a 16 bit value.

delayAfter-FirstCom-mand	Delay time to wait in msec after sending first commmand.
secondStar-tAddress	Address (MAC) of second command.
second-WordtoWrite	Second word to write.
delayAfter-Second-Command	Delay time to wait in msec after sending second commmand.
MACStyle	Style of MAC to write. Classic MAC or newMAC. Classic MAC = 0. NewMAC = 1. Any nonzero value will send NewMAC style.
protocol	Communication protocol used to send commands. SMB=0, I2C=1, HDQ=2
targetAd-dress	Targetaddress used to send commands. Not used with HDQ

Returns

Non zero if error or warning

4.1.2.23 EXPTYPE int SDKGetUSBAdapterVersion (const char ∗ *dataPipeName,* char ∗ *version*)

Read EV2X00 Firmware version. This may be used to test communication between host and adapter EV2300 and EV2400 use a different format for version. Version is returned char ∗. The buffer passed must be allocated with a length of greater than 10 bytes.

Parameters

dat-aPipeName	Name of pipe to use for communication
version	Buffer to return firmware version.

Returns

int Non zero if error or warning

4.1.2.24 EXPTYPE void SDKCloseDevice (const char ∗ *dataPipeName*)

Close Adapter Handle This will close adapter handle. Communication with the device continues but adapter is free to be reopened. The current adapter reported by CM will be a null string after this command is sent.

Parameters

dat-aPipeName	Name of data pipe

Returns

void

4.1.2.25 EXPTYPE int DiscoverNamesofCM (char ∗ *cmexecuting,* int ∗ *totalCm*)

Discover names and total number of CM processes currently executing on system. Command channel pipe server names are returned in a comma separated CString Recommended buffer length is 2000 bytes.

Parameters

cmexecuting	buffer to hold returned comma separated list of all executing CM command pipe names
totalCm	Total number of executing communication manager processes on system

Returns

Non zero if error or warning

4.1.2.26 EXPTYPE int DiscoverAllCMProcesses (int ∗ *totalCm*)

Returns number of running CM processes executing on system. This is done by probing each command channel.

Parameters

totalCm	Number of executing CM processes found on system

Returns

Non zero if error or warning

4.1.2.27 EXPTYPE int CreateChildProcess (char ∗ *cmdline,* int *startupMode*)

Start a new stand alone process with CreateProcess. Pass entire command line including arguments without -c option. Example command line is "c:\\tmp\\commmgr.exe" or "c:\\tmp\\commmgr.exe -l c:\\tmp\\logfile.txt" Process will start either in a console in the background.

Parameters

cmdline	cmdline of process to spawn
startupMode	Startup up mode 0=FOREGROUND 1=BACKGROUND

Returns

Microsoft windows error codes.

4.1.2.28 EXPTYPE int DoesCMUsingCmdChannelExist (const char * *commandChannelName*)

Test if an instance of commmgr.exe is executing using command channel pipe name.

Parameters

commandChannelName	Command channel pipe name to discover

Returns

1 if exist 0 if this name does not exist on system

4.1.2.29 TranslateCommError (int *errorCode*)

Translate a BMS tools communication error code This translates EV2X00 adapter, updater, and tools SDK error codes to text.

Parameters

errorCode	Error code to translate

Returns

Translated error code with english translation return in a CString

4.1.2.30 EXPTYPE void RegisterPIDwithCM (const char * *cmdPipeName*)

Register a process by sending process ID to CM. The CM keeps a record of all process IDs used by connection processes to determine when CM can terminate. A new process must register once with each instance of a CM that it will use. The CM will terminate when all registered process IDs no longer exist on system.

Parameters

cmdPipeName	of command pipe

Returns

void

4.1.2.31 EXPTYPE int TerminateCM (const char * *cmdPipeName*)

Terminate CM process using this command channel pipe name.

Parameters

cmdPipeName	Command Channel pipe name of process to terminate

Returns

Non zero if error or warning

4.1.2.32 EXPTYPE int SetupAppwithFirstFreeAdapter (char ∗ *cmdPipeName,* char ∗ *dataPipeName,* char ∗ *priorityPipeName,* char ∗ *evDeviceName,* char ∗ *exePathwithArgs,* int *buffersize,* int *processCreateStyle*)

Spawn a new commmgr.exe on an unused command channel. Open first free adapter. Prepare for EVM communication. Returns all pipe names and EV2X00 device name opened. CM process will spawn in background or foreground based on value of processCreateStyle. This is equivalant to how current tools obtain any free adapter handle This function will kill the created CM if no free adapters are available or any other error.

Parameters

cmdPipeName	Name of Command Channel pipe used by new CM process
dataPipeName	Name of data channel used by new CM process
priorityPipeName	Priority Channel pipe name used by new CM process
evDeviceName	EV2X00 device file name of acquired adapter
exePathwithArgs	Full path to commmgr.exe including any addtional command line options other than -c
buffersize	Minimum buffer size of all char passed char ∗ parameters. 256 bytes minimum
processCreateStyle	0=BACKGROUND 1=FOREGROUND (Console).

Returns

Non zero if error or warning

4.1.2.33 EXPTYPE int UseSpecificEV2X00 (const char ∗ *cmdPipeName,* const char ∗ *evDeviceName*)

Use specific EV2X00 device with a communication manager. This will close current device and use the device file name specified in parameter if it is free. An EV2400 adapter always uses the same HID device name. An EV2300 adapter always uses a different name.

Parameters

cmd-PipeName	Name of Command Channel pipe name to send command
evDevice-Name	Full path of EV2X00 adapter device name to use

Returns

Non zero if error or warning

4.1.2.34 EXPTYPE int IsCmdChannelNameValid (const char ∗ *cmdPipeName*)

Verify that command channel pipe name is a well known name.

Parameters

cmd-PipeName	Command Channel pipe name well known name. return TRUE (1) if a legal command channel name and name is not in use.

4.1.2.35 EXPTYPE int GetDataandPriorityNames (const char ∗ *cmdPipeName,* char ∗ *dataPipeName,* char ∗ *priorityPipeName,* int *bufferSize*)

Get data and priority pipe names for an executing communication manager process using this command channel name.

cmdPipeName must be a well known command channel special file name. Sixteen instance are supported. The well know command channel names are \\.\pipe\cm\cmdslot[0-15]

Parameters

cmd-PipeName	Name of command pipe to send to return other pipe names in use. This must be a well known command channel name.
dat-aPipeName	Name of data channel pipe used by this CM
priorityP-ipeName	Name of priority channel pipe used by this CM
bufferSize	Size of buffers passed. Recommended length is a minimum of 256 bytes.

Returns

Non zero if error or warning

A return code of 1235, "Command channel name is invalid", is returned if passed cmd-PipeName is not a well known name. A return code of 1233, "Command channel not found", is returned if cmdPipeName is a well know name be no communication manager on the system is using this name.

4.1.2.36 EXPTYPE int GetCMAdapterHandle (const char ∗ *cmdPipeName,* char ∗ *evDeviceName*)

Returns the name of adapter handle device file name used by an executing communication manager on command pipe name. If CM is executing but does not have a handle, parameter evDeviceName returned will be a null string. Use a minimum buffer length of 256 bytes.

Parameters

cmdPipeName	Command Channel name to query
evDeviceName	EV2X00 Adapter device name used by this CM instance

Returns

Non zero if error or warning

4.1.2.37 EXPTYPE int GetPriorityChannelState (const char ∗ *cmdPipeName,* int ∗ *status*)

This function queries and returns the state of the priority channel of an executing CM. Valid state are OFF=0, ON=1, CANCELLED=2.

Parameters

cmdPipeName	Command channel pipe name
status	Priority channel status This holds the returned status described above

Returns

Non zero if error or warning

4.1.2.38 EXPTYPE int StartPriorityChanSequence (const char ∗ *cmdPipeName*)

Begin a priority channel sequence of EVM commands. This will return an error if a priority sequence is already in progress. The data channel will be suspended until an EndPriorityChanSequence is sent to release this channel.

Parameters

cmdPipeName	Name of command channel pipe name

Returns

Non zero if error or warning

4.1.2.39 EXPTYPE int EndPriorityChanSequence (const char ∗ *cmdPipeName*)

End a priority channel sequence of EVM commands. This will terminate the priorty channel and release pending data channel operations in progress. Data channel requests will be serviced. CM returns to normal operation.

Parameters

cmd-PipeName	Command channel pipe name

Returns

Non zero if error or warning

4.1.2.40 EXPTYPE int SetPriorityChanKeepAlive (const char ∗ *cmdPipeName*)

Set priority keep alive. This keep alive is sent at a minimum of once a second for a long proiroty channel sequence. The priority channel will automatically cancel a priority sequence and return to normal operation if not refreshed with a keepalive packet. An example of is a long sequence of commands flashing a device. This command is required to be sent at least once per second to keep priority channel alive.

Parameters

cmd-PipeName	Command channel pipe name

Returns

Non zero if error or warning

4.1.2.41 EXPTYPE int GetSystemInventory (COMCHANNELINFO *cmStats[]*)

This scans the entire system and reports executing CM processes, channel names, adapter handles, and adapter device name for each unique command channel. All well known command channel names are probed. The result is return in an array of COMCHANNELINFO structures. The address of the structure defined as COMCHANNELINFO sysInv[MAXCMPROCESSES] is passed as the first parameter. Structure will be updated based on max number of CMs. Data, priority pipe name, and adapter device name will be null for a command pipe that is unassigned and available.

Parameters

cmStats[]	Array of COMCHANNELINFO structures to return results.

Returns

Non zero if error or warning

4.1.2.42 EXPTYPE int GetAllFreeAdapters (const char ∗ *cmdPipeName,* char ∗ *adapterBuffer*)

Get all free EV2X00 adapters connected and ready to use on a system. These are searched using an already executing CM. This returns a comma separated list of adapter free adapter device names. Buffer length to hold adapter names should be a minimum of 1900 bytes. This will report the first 16 free adapters found.

Parameters

cmd-PipeName	Command Channel pipe name
adapter-Buffer	Buffer to return results. A comma separated list of all free adapters device names is returned.

Returns

Non zero if error or warning

4.1.2.43 EXPTYPE int ResetCM (const char ∗ *cmdPipeName*)

Reset communication manager to initial state. This function will cancel a priority sequence if in progress and close adapter handle.

Parameters

cmd-PipeName	∗commandChannel Command channel pipe name.

Returns

Non zero if error or warning

4.1.2.44 EXPTYPE int GetCMReleaseVersion (const char ∗ *cmdPipeName,* char ∗ *versionString*)

Returns the release version of the communication manager in a version string. CString returned will not be greater than 40 bytes in length.

Parameters

cmd-PipeName	Command channel pipe name
version-String	Buffer to return version string. Format is X.X.X.X

Returns

Non zero if warning or error

4.1.2.45 EXPTYPE int DoesCMHaveHandle (const char ∗ *cmdPipeName*)

Does CM have an adapter handle.

Parameters

cmd-PipeName	Command channel pipe name

Returns

1=TRUE if has adapter handle. 0=FALSE if CM does not have an adapter handle.

4.1.2.46 SDKSetPinVoltage (const char ∗ *dataPipeName,* short *virtualPinNumber,* short *voltageLevelmV*)

This will enable output and set the pin voltage for EV2400 pins on VOUT1, VOUT2, VOUT3(not used), VOUT4 EV2400 : 1=VOUT1, 2=VOUT2, 3=VOUT3 not accessible, 4= VOUT4 (Voltage 3300) EV2400 FW is not working as of 3/13/2013 EV2300 : 1=VOUT, 2=SDA & SCL, 3=SDA & SCL , 4= HDQ (Voltage 5000 for EV2300 for high) EV2300 note: SDA and SCL lines are controlled together, so both will go high or low. New EV2300 packet is required to allow independent control, or the state information needs to be stored int this DLL which will cause problems if user disconnectes reconnects

Parameters

dat-aPipeName	Name of data pipe to use for communications
virtualPin-Number	Name of virtual pin to change
volt-ageLevelmV	Voltage level. Use 5000 for EV2300. EV2400 has adjustable levels need FW work.

Returns

int Non zero for errors and warnings

4.1.2.47 SDKGetBoardType (const char ∗ *dataPipeName,* unsigned char ∗ *boardType,* unsigned short ∗ *numBytesRead*)

This will return the EV2400 board type

Parameters

dat-aPipeName	Name of data pipe to use for communications
boardType	Buffer to return board type
numBytes-Read	short ∗ Number of bytes in the board type

Returns

int Non zero for errors and warnings

4.1.2.48 SDKSetVoutWithTimeout (const char ∗ *dataPipeName,* unsigned char *vout1Enable,* unsigned char *vout2Enable,* unsigned char *vout3Enable,* unsigned char *vout4Enable*)

/∗∗ This will set the EV2400 VOUT pins with timeout

Parameters

dataPipeName	Name of data pipe to use for communications
vout1Enable	Enable setting for VOUT1 (CHG_EN)
vout2Enable	Enable setting for VOUT2 (LOAD_EN)
vout3Enable	Enable setting for VOUT3 (CHGR1_EN)
vout4Enable	Enable setting for VOUT4 (CHGR2_EN)

Returns

int Non zero for errors and warnings

4.1.2.49 SDKSetPWMConfig (const char ∗ *dataPipeName,* unsigned short *dutyCycle*)

Set PWM Configuration.

/∗∗

Parameters

dataPipeName	Name of data pipe to use for communications
dutyCycle	Duty Cycle of the PWM)

Returns

int Non zero for errors and warnings

4.1.2.50 int SendRecCommand (const char ∗ *dataPipeName,* int *cmd,* char ∗ *messagetoSend,* char ∗ *messageRecv*)

Send and receive a command packet to a command pipe server. Each command packet has a unique format. The command packet format description is in the Communication Manager design document. This is primarily for testing. Message format as defined in COMMESSAGE type Both sent message and returned message will be in COMMESSAGE format A buffer size of 1900 bytes is recommended. This is used for debugging.

Parameters

dat-aPipeName	Name of command pipe to send command
cmd	Number of command.
message-toSend	Message to send in packet payload
mes-sageRecv	buffer to hold returned from command

Returns

Non zero if error or warning

4.1.2.51 int IsDataPipeNameValid (const char ∗ *dataPipeName*)

This will check the name of the datapipe and attempt to ensure it is valid. This will return a 1 of the name is valid or a zero if it is not.

Parameters

dat-aPipeName	Name of data pipe name to check.

Returns

int Non zero for errors and warnings

4.1.2.52 EXPTYPE int C816Senc (const char ∗ *dataPipeName,* const char ∗ *srecFileName,* int *operation,* int *protocol*)

The will program or create a C816 senc from connected device. These are generally single cell devices like bq27541. ROM version is not checked. Device must be executing ROM prior to call. SMB supported family is bq8030.

Parameters

dat-aPipeName	Name of of data pipe to use.
srecFile-Name	Name of senc or srecord file to program.
operation	Operation = 0 will flash the senc file. Operation = 1 will create an senc from C816 connected device.
protocol	0=SMBUS 2=I2C. SMBUS is not implemented.

Returns

Non zero for errors and warnings

4.1.2.53 EXPTYPE int ProgramC816Senc_I2C (const char ∗ *dataPipeName,* const char ∗ *srecFileName*)

4.1.2.54 EXPTYPE int SDKGetEV2300Name (const char ∗ *dataPipeName,* char ∗ *ev2300Name*)

Get name stored in an EV2300 communications adapter. This will return a previously stored name in EV2300 data flash that is used as an identifier. The expected length of this name will be 32 bytes or less. A pointer to a pre-allocated byte buffer of at least 33 bytes in length is passed. A CString with the EV2300 name will reside in this buffer after the call. Use the function SDKSetEV2300Name to write a name into an EV2300. This function does nothing if adapter is an EV2400. Each EV2400 has a unique serial number as shipped that can be used a unique identifier. This serial number is also available in the Windows Device Manager under Human Interface Devices HID-compliant device for the adapter. The "Parent" property contains this serial number. EV2400 version 0.27 and later firmware allows setting a unique name also in EV2400 flash. If a name is set, it overrides the serial number. Bq80xrw.dll version 0.1.69 and later allows EV2400 serial number to be returned in a hexadecimal string with the function SDKGetEV2300Name if name has been set using another tool. An example of an EV2400 serial number is 7624996F27002A00. The Parent propety in the device manager for this device is USB\VID_0451&PID_0037\7624996F27002A00.

Parameters

dat-aPipeName	Name of of data pipe to use.
ev2300Name	Byte buffer with name after function call.

Returns

Non zero for errors and warnings

4.1.2.55 EXPTYPE int SDKSetEV2300Name (const char ∗ *dataPipeName,* char ∗ *ev2300Name*)

Set EV2300 name writes a string to data flash inside and EV2300 communication adapter. This is used as an identifier. This name will be truncated if larger than 31 bytes. Names cannot be stored in an EV2400 adapter. Each EV2400 has a unique serial number based on die ID that will not change. This is the EV2400 identifier.

When this function is called with an EV2400 connected, the serial number identifier is returned in parameter ev2300Name. An example of an EV2400 serial number is D03D8A461B000E00. This serial number is available in Windows Device Manager under Human Interface Devices, HID-compliant device for the adapter. The property Parent contains the serial number after Changes to EV2400 firmware allow the setting of EV2400 name.

In both cases, the return code is zero if either adapter is found. The return code will be none zero if an adapter is not found.

Parameters

dat-aPipeName	Name of of data pipe to use.
ev2300Name	Name to write to EV2300 or serial number returned if EV2400. This parameter must be pre-allocated to a minimum of 32 bytes.

Returns

Non zero for communication errors and warnings.

4.1.2.56 EXPTYPE int SDKSetAdapterI2CClockSpeed (const char ∗ *dataPipeName*, int *speed*)

Set EV2300 or EV2400 I2C clock speed.

Parameters

dat-aPipeName	Name of of data pipe to use.
speed	0 for low speed. Any other value is high speed.

Returns

Non zero for communication errors and warnings.

4.1.2.57 EXPTYPE void setBMDKConstants (short *extPort*, int *isExtPort*, short *intPort*, int *isIntPort*, int *ocHandle*, int *isHandle*)

Sets BMDK Configuration parameters.

Parameters

extPort	BMDK External Port.
isExtPort	non-zero for true.
intPort	BMDK Internal Port.
isIntPort	non-zero for true.
ocHandle	One Controller Handle.
isHandle	non-zero for true.

Returns

void

4.1.2.58 EXPTYPE int SDKSetEV2400CommClkSpeed (const char ∗ *dataPipeName*, short *clkSpeedkHz*, short *tgtProtocol*)

Set EV2400 protocol clock speed for SMB or I2C in kHz.

Parameters

dat-aPipeName	Name of of data pipe to use.
clkSpeed-kHz	Clock speed in kHz.
tgtProtocol	SMB=0 I2C=1. Do not use for SPI or SPICRC.

Returns

Zero for success or non-zero on error

4.1.2.59 EXPTYPE int SDKSPIConfigure (const char ∗ *dataPipeName,* short *polarity,* short *phase,* int *clkSpeedkHz,* short *SPICRCEnabled*)

Configure SPI EV2400 master characteristics. EV2400 with firmware version 0.30 or later is required.

Parameters

dat-aPipeName	Name of of data pipe to use.
polarity	SPI polarity. Note: bq769x2 -A2 devices only support CPOL=0 and CPHA=0. polarity=0 CPOL=0(Clock idles at 0) polarity=1 CPOL=1(Clock idles at 1)
phase	SPI phase. phase=0 CPHA=0(Half cycle of clock idle followed by a half cycle of clock asserted. phase=1 CPHA=1(Half cycle of clock asserted followed by a half cycle of clock idle.
clkSpeed-kHz	in kHz. SPI bit rate in kHz. 2MHz maximium. The specific clock frequency are 2MHz, 1Mhz, 500kHz, 250kHz, 125kHz. ... EV2400 clock is calculated as 4Mhz/(n << 1).
SPI-CRCEnabled	0=disabled 1=enabled.

Returns

Zero for success or non-zero on error.

4.1.2.60 EXPTYPE int SDKSPIReadConfiguration (const char ∗ *dataPipeName,* short ∗ *polarity,* short ∗ *phase,* int ∗ *clkSpeedkHz,* short ∗ *SPICRCEnabled*)

Read EV2400 SPI master characteristics. EV2400 with firmware version 0.30 or later is required.

Parameters

dat-aPipeName	Name of of data pipe to use.

polarity	Configured SPI polarity. Note: bq769x2 -A2 devices only support CPOL=0 and CPHA=0. polarity=0 CPOL=0(Clock idles at 0) polarity=1 CPOL=1(Clock idles at 1)
phase	Configured SPI phase. phase=0 CPHA=0(Half cycle of clock idle followed by a half cycle of clock asserted. phase=1 CPHA=1(Half cycle of clock asserted followed by a half cycle of clock idle.
clkSpeed-kHz	in kHz. SPI bit rate in kHz. 2MHz maximium. The specific clock frequency are 2MHz, 1Mhz, 500kHz, 250kHz, 125kHz. ... EV2400 clock is calculated as 4Mhz/(n $<<$ 1).
SPI-CRCEnabled	0=disabled 1=enabled.

Returns

Zero for success or non-zero on error.

4.1.2.61 EXPTYPE int SDKSPI16ReadBlock (const char ∗ *dataPipeName,* short *startAddress,* unsigned char ∗ *dataRead,* short *numBytestoRead*)

Read a SPI16 Block. EV2400 with firmware version 0.30 or later is required.

Parameters

dat-aPipeName	Name of data pipe to use.
startAddress	This is the start address of the data to read.
dataRead	Preallocated buffer to return bytes read from device.
numByte-stoRead	Number of bytes to read.

Returns

Zero for success or non-zero on error.

4.1.2.62 EXPTYPE int SDKSPI16WriteBlock (const char ∗ *dataPipeName,* short *startAddress,* unsigned char ∗ *datatoWrite,* short *numBytestoWrite,* short *delayAfterWrite*)

Write a SPI16 Block. EV2400 with firmware version 0.30 or later is required.

Parameters

dat-aPipeName	Name of of data pipe to use.
startAddress	This is the start address of the data to read.
datatoWrite	Bytes to write to device.
num-BytestoWrite	Number of bytes to write.

delayAfter-Write	Delay in msec after write.

Returns

Zero for success or non-zero on error.

4.1.2.63 EXPTYPE int SDKSPI16WriteBlkReadBlk (const char ∗ *dataPipeName,* short *firstBlkStartAddress,* unsigned char ∗ *firstBlkData,* short *firstNumBytestoWrite,* short *delayAfterFirstBlock,* short *block2StartAddress,* \unsigned char ∗ *dataRead,* short *numBytestoRead*)

This is used to send a MAC subcommmand and read block of data return by subcommand. SPI Write first block with number of bytes specified with delay in msec after sending first block. Then read a block of of data of specified length. Length must be less than 128 bytes for EV2400. Write data can be up to 4 bytes but only 2 are required for a MAC subcommand. Delay after write is used to give MAC command time to execute. These commands are atomic. Both will be executed and cannot be interrupted by another thread.

Parameters

dat-aPipeName	Name of data pipe
firstBlkStar-tAddress	Start address of first block to to write.
firstBlkData	First block of data. The can be 1, 2, 3, or 4 bytes in length
firstNum-BytestoWrite	Number of bytes to write. This must be a number from 1 to 4.
delayAfter-FirstBlock	Delay in msec after sending first block before beginning to send second block
block2StartAc	Start address of block to read.
dataRead	buffer to store returned bytes read from device
numByte-stoRead	Number of bytes read from second block

Returns

Non zero if error or warning from read block command

4.1.2.64 EXPTYPE int SPIAA_Control (const char ∗ *dataPipeName,* int *operation,* unsigned int *serialNumber*)

Internal function to control Aardvark SPI for debugging.

Parameters

dat-aPipeName	Name of of data pipe to use.
operation	The defined operations are as follows. operation=0 Open Aardvark by serial number. setup default clock, phase, and polarity Set SPI bitRate to 1MHz

operation=1 Close Aardvark using current handle.

operation=2 return current Aardvark handle

operation=3 Set clock(bitrate) to 1 MHz

operation=4 Set clock(bitrate) to 2 MHz

operation=5 Set clock(bitrate) to 400 kHz

Parameters

serialNum-ber	Last 6 digits of Aardvark serial number to connect. Example Aardvark serial number is 2237-476036 to setup. This parameter will be = 476036

Returns

Aardvark handle. A negative number is an Aardvark error

4.1.2.65 EXPTYPE int SDKUseAardvarkSPI (const char ∗ *dataPipeName,* int *flag*)

Index