SPECIFICATION: QCOM INTERFACE DLL

Rev	ECO	Date	Change By	Description
A0	EC1500	09/27/2010	D. Sealey	Initial release
B0	EC2183	04/06/2011	Noji Ratzlaff	Added many new functions, including firmware update controls
B1	EC2692	05/03/2011	Noji Ratzlaff	Added a table of contents; added a new transducer type (0x05) to ***QD_GetTransducerType*** and ***QD_GetUnitStatusRegister***; and added extra error codes to ***QD_GetTransducerCurrent*** and ***QD_GetTransducerVoltage***
B2	EC2793	07/18/2011	Noji Ratzlaff	Added ***QD_GetMemoryType*** to accommodate changes in QCOM firmware 1.2 to correctly detect the memory type of the target device. Also added example C code for each function.
B3	EC3186	01/17/2012	Noji Ratzlaff	Changed ***QD_GetProductString*** to ***QD_GetProductInfo*** and ***QD_GetI2CBaudRate*** to ***QD_GetI2CDataRate*** and ***QD_SetI2CBaudRate*** to ***QD_SetI2CDataRate*** and ***QD_GetErrorCode*** to ***QD_GetInternalErrorCode*** and ***QD_ClearError*** to ***QD_ClearInternalError***. Enhanced the meaning and composition of the status returned from each function that returns an error code.
B4	EC3644	09/21/2012 11/03/2012	Noji Ratzlaff	Added a test in ***QD_ReadCoefficientDataFromHexFile*** for valid coefficient data. Changed the names of the following functions to more accurately reflect module-specific operations: QD_GetNumberOfUnits => QD_GetNumberOfModules QD_GetUnitControlRegister => QD_GetModuleControlRegister QD_GetUnitSerialNumber => QD_GetModuleSerialNumber QD_GetUnitFirmwareID => QD_GetModuleFirmwareID QD_GetUnitStatusRegister => QD_GetModuleStatusRegister QD_GetUnitModeSwitchSetting => QD_GetModuleModeSwitchSetting QD_ReadCoefficientDataFromUnitPage => QD_ReadCoefficientDataFromModulePage QD_ResetUnit => QD_ResetModule QD_SetUnitControlRegister => QD_SetModuleControlRegister QD_WriteCoefficientDataToUnitPage => QD_WriteCoefficientDataToModulePage

SPECIFICATION: QCOM INTERFACE DLL

1. SCOPE:

This document describes the custom DLL library functions (`qdUSB.dll`) used by the QCOM software and custom host applications that use the QCOM hardware.

2. DEFINITIONS, SPECIAL NOTES, MISCELLANEOUS:

2.1. Quartzdyne's custom DLL (`qdUSB.dll`) references functions in the Silicon Labs USB-microcontroller DLL (`SiUSBXp.dll`) and contains all custom functionality necessary for interfacing with one to sixteen QCOM modules and attached transducers. For QCOM interfacing, the application developer need only perform the function calls included in `qdUSB.dll`.

2.2. In general, the application initiates communication with the target module(s) by calling ***QD_GetNumberOfModules***, which will return the number of attached target modules. This value is then used as a range when calling ***QD_GetModuleSerialNumber*** and other functions to build a list of modules and their attributes.

To communicate with a QCOM module, the host must first call ***QD_Open*** using a zero-relative index determined from the call to ***QD_GetNumberOfModules***. ***QD_Open*** will return a handle to the module that is used in subsequent Read and Write accesses. When I/O operations are complete, the host terminates all communications with the module by calling ***QD_Close***.

2.3. The terms *unit*, *module*, *device*, and *pod* are synonymous in this document, and each refers to the physical piece of hardware represented by name of QCOM.

2.4. *In:* refers to a function parameter that requires information. *Out:* refers to a function parameter that provides information. *In/Out:* refers to a function parameter that both requires information at the time of the function call and provides information at the conclusion of the function call.

2.5. All references to PSI are actually PSIA units, since Quartzdyne transducers are calibrated to absolute pressure.

2.6. This document is posted at http://qd.quartzdyne.com/public/QCOM/QCOM-Download.php

2.7. The error codes (***Return Value***) listed reflect only the lower WORD (16 bits) of the 32-bit status DWORD returned by each function that returns its status. The upper BYTE (bits 24 – 31) of the status DWORD is the function number. The next BYTE (bits 16 – 23) is the problem locus, indicating the approximate and relative location within the function where the error was detected. See section 4 for details.

SPECIFICATION: QCOM INTERFACE DLL

3. FUNCTION DESCRIPTIONS

SPECIFICATION: QCOM INTERFACE DLL

3.1. QD_GetNumberOfModules 0x1E

Description	Returns the number of QCOM modules attached to the host
Prototype	`DWORD QD_GetNumberOfModules(void);`
Parameters	• None
Return Value	0 *No modules are found* Nonzero *Number of modules attached*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD numberOfModules;`
	`numberOfModules = QD_GetNumberOfModules();`
	`printf("%d QCOM modules are attached\n", numberOfModules);`
	Output:
	`2 QCOM modules are attached`

SPECIFICATION: QCOM INTERFACE DLL

3.2. QD_GetProductInfo 0x20

Description	Returns a null-terminated product-related string of the selected information

Prototype	`DWORD QD_GetProductInfo(`
	`  DWORD unitNumber,`
	`  LPBYTE productInfoString,`
	`  DWORD item);`

Parameters	• *unitNumber*— In: Zero-relative module index, for the module whose product description string or serial number string is requested. The index for the first module is 0, and the index for the last module is the value returned by ***QD_GetNumberOfModules***, minus 1. • *productInfoInfo*— Out: 256-byte buffer to store the device description, serial number string, etc. • *item*— In: Value that determines whether *productInfoString* contains a serial number, product description, manufacturer name, Vendor ID, or Product ID string, and is one of the following: *QD_RETURN_SERIAL_NUMBER* 0 *QD_RETURN_DESCRIPTION* 1 *QD_RETURN_COMPANY* 2 *QD_RETURN_VID* 3 *QD_RETURN_PID* 4
Return Value	`0x0000` *QD_SUCCESS* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x000A` *QD_ERROR_FUNCTION_NOT_SUPPORTED* `0x00FF` *QD_ERROR_NO_DEVICE_FOUND*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	`DWORD unitNumber = 2;`
	`char productString[QD_MAXIMUM_PRODUCT_STRING_SIZE];`
	
	`status = QD_GetProductInfo(`
	`  unitNumber,`
	`  (LPBYTE) productString,`
	`  QD_RETURN_DESCRIPTION);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The product name of module %d is %s\n",`
	`  unitNumber,`
	`  productString);`
	`}`
	
	Output:
	
	`The product name of module 2 is QCOM Module 02`

3.3. QD_Open 0x2D

Description	Opens and returns a handle to the module that will be used for subsequent accesses

Prototype	`DWORD QD_Open(`
	`  DWORD unitNumber,`
	`  HANDLE *unitHandle);`

Parameters	• *unitNumber*— In: Zero-relative module index. See ***QD_GetProductString*** for a more complete description. • *unitHandle*— Out: Pointer to the module handle, to be used by subsequent I/O accesses to the module
Return Value	`0x0000` *QD_SUCCESS* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x00FF` *QD_ERROR_NO_DEVICE_FOUND*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	`DWORD unitNumber = 0;`
	`HANDLE unitHandle;`
	
	`status = QD_Open(`
	`  unitNumber,`
	`  &unitHandle);`

3.4. QD_Close 0x14

Description	Closes an open module using the handle provided by ***QD_Open***
Prototype	`DWORD QD_Close(` `  HANDLE unitHandle);`
Parameters	• *unitHandle*—In: Module handle returned by ***QD_Open***
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	
	`status = QD_Close(unitHandle);`

3.5. QD_SetTimeouts 0x3B

Description	Sets the Read and Write operation timeouts, which are used for Read and Write operations that are called synchronously. The default value for timeouts is 0xFFFFFFFF (infinite), but they can be set for any finite number of milliseconds between 0x00000000 and 0xFFFFFFFE, inclusive.

Prototype	`DWORD QD_SetTimeouts(`
	`  DWORD readTimeout,`
	`  DWORD writeTimeout);`

Parameters	• *readTimeout*— In: Read operation timeout in milliseconds • *writeTimeout*— In: Write operation timeout in milliseconds
Return Value	0x0000 *QD_SUCCESS* 0x0006 *QD_ERROR_INVALID_PARAMETER*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	
	`status = QD_SetTimeouts(500, 1000);`

3.6. QD_GetTimeouts 0x21

Description	Returns the current Read and Write timeouts. If a timeout value is 0xFFFFFFFF (infinite) it has been set to wait indefinitely; otherwise, the timeouts are specified in milliseconds.

Prototype	`DWORD QD_GetTimeouts(`
	`  LPDWORD readTimeout,`
	`  LPDWORD writeTimeout);`

Parameters	• *readTimeout*— Out: Read operation timeout in milliseconds • *writeTimeout*— Out: Write operation timeout in milliseconds
Return Value	0x0000 *QD_SUCCESS* 0x0006 *QD_ERROR_INVALID_PARAMETER*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD readTimeout;`
	`DWORD writeTimeout;`
	`DWORD status;`
	
	`status = QD_GetTimeouts(`
	`  (LPDWORD) &readTimeout,`
	`  (LPDWORD) &writeTimeout);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The read timeout is %d ms and the write timeout is %d ms\n",`
	`  readTimeout, writeTimeout);`
	`}`
	
	Output:
	
	`The read timeout is 500 ms and the write timeout is 1000 ms`

SPECIFICATION: QCOM INTERFACE DLL

3.7. QD_CheckReceiveQueue 0x12

Description	Returns the number of bytes in the Receive Queue and a status value that indicates whether an Overrun has occurred and whether the Receive Queue is ready for reading. Upon encountering an Overrun condition, it is recommended that data transfer be stopped and all buffers flushed using ***QD_FlushBuffers***.

Prototype	`DWORD QD_CheckReceiveQueue(`
	`  HANDLE unitHandle,`
	`  LPDWORD numberOfBytesInQueue,`
	`  LPDWORD queueStatus);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *numberOfBytesInQueue*— Out: Pointer to the number of bytes currently in the Receive Queue • *queueStatus*— Out: Pointer to one or more of the following: 0x00 *QD_QUEUE_NO_OVERRUN* 0x00 *QD_QUEUE_EMPTY* 0x01 *QD_QUEUE_OVERRUN* 0x02 *QD_QUEUE_READY*
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0008 *QD_ERROR_DEVICE_IO_FAILED*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD numberOfBytes;`
	`DWORD queueStatus;`
	`DWORD status;`
	
	`status = QD_CheckReceiveQueue(`
	`  unitHandle,`
	`  (LPDWORD) &numberOfBytes,`
	`  (LPDWORD) &queueStatus);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("There are %d bytes in the queue, and the queue status is %d\n",`
	`  numberOfBytes, queueStatus);`
	`}`
	`if (queueStatus & QD_QUEUE_OVERRUN)`
	`{`
	`  QD_FlushBuffers(unitHandle, 1, 1);`
	`}`
	
	Output:
	
	`There are 8 bytes in the queue, and the queue status is 2`

SPECIFICATION: QCOM INTERFACE DLL

3.8. QD_FlushBuffers 0x17

Description	Flushes both the Receive Buffer in the device driver and the Transmit Buffer in the USB driver. **Note:** *flushTransmitBuffer* and *flushReceiveBuffer* have no effect on the QCOM module.

Prototype	`DWORD QD_FlushBuffers(`
	`  HANDLE unitHandle,`
	`  BYTE flushTransmitBuffer,`
	`  BYTE flushReceiveBuffer);`

Parameters	• *unitHandle*—In: Module handle returned by ***QD_Open*** • *flushTransmitBuffer*— In: Flushes the internal Transmit Buffer if nonzero • *flushReceiveBuffer*— In: Flushes the Receive Buffer if nonzero
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD numberOfBytes;`
	`DWORD queueStatus;`
	`DWORD status;`
	
	`QD_CheckReceiveQueue(`
	`  unitHandle,`
	`  (LPDWORD) &numberOfBytes,`
	`  (LPDWORD) &queueStatus);`
	`if (queueStatus & QD_QUEUE_OVERRUN)`
	`{`
	`  status = QD_FlushBuffers(unitHandle, 1, 1);`
	`}`

SPECIFICATION: QCOM INTERFACE DLL

3.9. QD_GetTransducerType 0x24

Description	Used to read the type of transducer connected to the QCOM module

Prototype	`DWORD QD_GetTransducerType(`
	`  HANDLE unitHandle,`
	`  LPBYTE transducerType);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • transducerType— Out: Pointer to the transducer type, as follows: `0x00` No transducer detected `0x01` Analog / frequency transducer detected `0x02` Pre-FireFly digital transducer detected `0x03` FireFly digital transducer (with VREF) detected `0x04` FireFly digital transducer (without VREF) detected `0x05` Digital transducer without memory detected
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE transducerType;`
	`DWORD status;`
	
	`status = QD_GetTransducerType(`
	`  unitHandle,`
	`  (LPBYTE) &transducerType);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The attached transducer is type %d\n",`
	`  transducerType);`
	`}`
	
	Output:
	
	`The attached transducer is type 4`

SPECIFICATION: QCOM INTERFACE DLL

3.10. QD_GetTransducerCounts 0x22

Description	Returns both the pressure and temperature counts from the specified transducer

Prototype	`DWORD QD_GetTransducerCounts(`
	`  HANDLE unitHandle,`
	`  LPDWORD pressureCount,`
	`  LPDWORD temperatureCount);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *pressureCount*— Out: Pointer to the pressure (Xp) count result • *temperatureCount*— Out: Pointer to the temperature (Xt) count result
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING* `0x0011` *QD_ERROR_XD_PRESSURE_COUNT_ZERO* `0x0012` *QD_ERROR_XD_TEMPERATURE_COUNT_ZERO* `0x0013` *QD_ERROR_XD_BOTH_COUNTS_ZERO*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD pressureCount;`
	`DWORD temperatureCount;`
	`DWORD status;`
	
	`status = QD_GetTransducerCounts(`
	`  unitHandle,`
	`  (LPDWORD) &pressureCount,`
	`  (LPDWORD) &temperatureCount);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The pressure count is %d and the temperature count is %d\n",`
	`  pressureCount, temperatureCount);`
	`}`
	
	Output:
	
	`The pressure count is 10273309 and the temperature count is 36017925`

3.11. QD_CalculatePressureOrTemperature 0x11

Description	Calculates either the pressure in PSI or the temperature in Celsius, given both pressure and temperature count values, and the 256-byte buffer of coefficient data

Prototype	`DWORD QD_CalculatePressureOrTemperature(`
	`  BYTE PorT,`
	`  LPBYTE coefficientData,`
	`  DWORD pressureCount,`
	`  DWORD temperatureCount,`
	`  DOUBLE *result);`

Parameters	• *PorT*— In: Value that determines the calculation result as either Pressure or Temperature, as follows: *QD_CALCULATE_PRESSURE_PSI* 0 *QD_CALCULATE_TEMPERATURE_CELSIUS* 1 • *coefficientData*— In: 256-byte buffer of coefficient file data • *pressureCount*— In: The pressure (Xp) count value used in the calculation • *temperatureCount*— In: The temperature (Xt) count value used in the calculation • *result*— Out: Pointer to the calculation result
Return Value	`0x0000` *QD_SUCCESS*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD pressureCount;`
	`DWORD temperatureCount;`
	`DWORD status;`
	
	`status = QD_GetTransducerCounts(`
	`  unitHandle,`
	`  (LPDWORD) &pressureCount,`
	`  (LPDWORD) &temperatureCount);`
	`if (status == QD_SUCCESS)`
	`{`
	`  BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`  DOUBLE temperature;`
	
	`  QD_ReadCoefficientDataFromDevice(`
	`  unitHandle,`
	`  (LPBYTE) coefficientData);`
	`  status = QD_CalculatePressureOrTemperature(`
	`  QD_CALCULATE_TEMPERATURE_CELSIUS,`
	`  (LPBYTE) coefficientData,`
	`  pressureCount,`
	`  temperatureCount,`
	`  &temperature);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  printf("The current transducer temperature is %0.2f °C\n",`
	`  temperature);`
	`  }`
	`}`
	
	Output:
	
	`The current transducer temperature is 24.04 °C`

SPECIFICATION: QCOM INTERFACE DLL

3.12. QD_GetPressureAndTemperature 0x1F

Description	Calculates the pressure in PSI and temperature in Celsius by calling ***QD_GetTransducerCounts*** and ***QD_CalculatePressureAndTemperature*** in a one-call-does-all function

Prototype	`DWORD QD_GetPressureAndTemperature(`
	`  HANDLE unitHandle,`
	`  LPBYTE coefficientData,`
	`  DOUBLE *pressurePSI,`
	`  DOUBLE *temperatureCelsius);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *coefficientData*— In: 256-byte buffer of coefficient file data • *pressurePSI*— Out: Pointer to the pressure value in PSI • *temperatureCelsius*— Out: Pointer to the temperature value in Celsius
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING* 0x0011 *QD_ERROR_XD_PRESSURE_COUNT_ZERO* 0x0012 *QD_ERROR_XD_TEMPERATURE_COUNT_ZERO* 0x0013 *QD_ERROR_XD_BOTH_COUNTS_ZERO*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	
	`status = QD_ReadCoefficientDataFromDevice(`
	`  unitHandle,`
	`  (LPBYTE) coefficientData);`
	`if (status == QD_SUCCESS)`
	`{`
	`  DOUBLE pressure;`
	`  DOUBLE temperature;`
	
	`  status = QD_GetPressureAndTemperature(`
	`  unitHandle,`
	`  (LPBYTE) coefficientData,`
	`  &pressure,`
	`  &temperature);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  printf("The pressure is %0.2f PSI and the temperature is %0.2f °C\n",`
	`  pressure, temperature);`
	`  }`
	`}`
	
	Output:
	
	`The pressure is 13.78 PSI and the temperature is 24.04 °C`

SPECIFICATION: QCOM INTERFACE DLL

3.13. QD_ReadCoefficientDataFromSourcePage 0x31

Description	Returns a 256-byte buffer of coefficient data retrieved from either the module memory or digital transducer memory at the specified memory page

Prototype	`DWORD QD_ReadCoefficientDataFromSourcePage(`
	`  HANDLE unitHandle,`
	`  BYTE device,`
	`  BYTE pageNumber,`
	`  LPBYTE coefficientData);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *device*— In: Source of the coefficient data, as follows: *QD_DEVICE_TRANSDUCER* 0x00 *QD_DEVICE_MODULE* 0x01 • *pageNumber*— In: Offset of the 256-byte memory page from which to read, as follows: 0x00 Page at offset 0x0000 (page 0) 0x01 Page at offset 0x0100 (page 1) 0x02 Page at offset 0x0200 (page 2) . . . 0x3F Page at offset 0x3F00 (page 63) *pageNumber* must be a value between 0 and 63, inclusive • *coefficientData*— Out: 256-byte buffer to store the coefficient file data
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING* ‡ See the exception note for ***Next Byte*** in section 4

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	`status = QD_ReadCoefficientDataFromSourcePage(`
	`  unitHandle,`
	`  QD_DEVICE_TRANSDUCER,`
	`  0,`
	`  (LPBYTE) coefficientData);`

SPECIFICATION: QCOM INTERFACE DLL

3.14. QD_WriteCoefficientDataToTargetPage 0x42

Description	Writes a 256-byte buffer of coefficient data to either the module memory or the digital transducer memory at the specified memory page

Prototype	`DWORD QD_WriteCoefficientDataToTargetPage(`
	`  HANDLE unitHandle,`
	`  BYTE device,`
	`  BYTE pageNumber,`
	`  LPBYTE coefficientData);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *device*— In: Target of the coefficient data, as follows: *QD_DEVICE_TRANSDUCER* 0x00 *QD_DEVICE_MODULE* 0x01 • *pageNumber*— In: Offset of the 256-byte memory page to store the data, as follows: 0x00 Page at offset 0x0000 (page 0) 0x01 Page at offset 0x0100 (page 1) 0x02 Page at offset 0x0200 (page 2) . . . 0x3F Page at offset 0x3F00 (page 63) *pageNumber* must be a value between 0 and 63, inclusive • *coefficientData*— In: 256-byte buffer containing the coefficient file data
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000F *QD_ERROR_IO_PENDING* 0x0028 *QD_ERROR_WRITE_ACKNOWLEDGE_FAILED* 0x000E *QD_ERROR_NULL_POINTER_PARAMETER* ‡ See the exception note for ***Next Byte*** in section 4

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	`status = QD_ReadCoefficientDataFromHexFile(`
	`  (LPBYTE) "XD242109.hex",`
	`  (LPBYTE) coefficientData);`
	`if (status == QD_SUCCESS)`
	`{`
	`  status = QD_WriteCoefficientDataToTargetPage(`
	`  unitHandle,`
	`  QD_DEVICE_TRANSDUCER,`
	`  2,`
	`  (LPBYTE) coefficientData);`
	`}`

SPECIFICATION: QCOM INTERFACE DLL

3.15. QD_ReadCoefficientDataFromHexFile 0x30

Description	Retrieves a 256-byte buffer of coefficient data from the specified hex file path

Prototype	`DWORD QD_ReadCoefficientDataFromHexFile(`
	`  LPBYTE pathName,`
	`  LPBYTE coefficientData);`

Parameters	• *pathName*—In: Null-terminated string that specifies the path and filename of the coefficient data hex file • *coefficientData*— Out: 256-byte buffer to store the coefficient file data converted from the hex file
Return Value	0x0000 *QD_SUCCESS* 0x0014 *QD_ERROR_MEMORY_ALLOCATION_FAILED* 0x0015 *QD_ERROR_FILE_OPEN_FAILURE* 0x0017 *QD_ERROR_FILE_SIZE_MISMATCH* 0x0018 *QD_ERROR_INVALID_HEX_FILE_DATA* 0x0019 *QD_ERROR_INCORRECT_DATA_SIZE* 0x002A *QD_ERROR_NULL_POINTER_PARAMETER* 0x002B *QD_ERROR_ZERO_LENGTH_STRING_PARAMETER* 0x0040 *QD_ERROR_INVALID_COEFFICIENT_DATA* 0x0070 *QD_ERROR_FILE_NOT_FOUND*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	
	`status = QD_ReadCoefficientDataFromHexFile(`
	`  (LPBYTE) "XD242109.hex",`
	`  (LPBYTE) coefficientData);`
	`if (status == QD_SUCCESS)`
	`{`
	`  status = QD_WriteCoefficientDataToTargetPage(`
	`  unitHandle,`
	`  QD_DEVICE_TRANSDUCER,`
	`  2,`
	`  (LPBYTE) coefficientData);`
	`}`

SPECIFICATION: QCOM INTERFACE DLL

3.16. QD_ExecuteI2CCommand 0x16

Description	Explicitly reads or writes data on the I2C bus by means of a formatted command string

Prototype	`DWORD QD_ExecuteI2CCommand(`
	`  HANDLE unitHandle,`
	`  LPBYTE commandString,`
	`  LPBYTE replyString);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *commandString*— In: Null-terminated string that contains a properly formatted I2C command as described in the table on the following page • *replyString*— Out: Buffer large enough to store the null-terminated I2C response as described in the table of the following page
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE replyString[QD_MAXIMUM_TRANSFER_SIZE];`
	`DWORD status;`
	
	`status = QD_ExecuteI2CCommand(`
	`  unitHandle,`
	`  (LPBYTE) "S9DFFFFFFFFP",`
	`  (LPBYTE) replyString);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The pressure count is 0x%c%c%c%c%c%c%c%c\n",`
	`  replyString[3], replyString[4], replyString[5],`
	`  replyString[6], replyString[7], replyString[8],`
	`  replyString[9], replyString[10]);`
	`}`
	
	Output:
	
	`The pressure count is 0x009CC1EA`

SPECIFICATION: QCOM INTERFACE DLL

QD_ExecuteI2CCommand cont'd

The command string and response string are defined as follows:

- 'S' is I2C *start*, 'R' is I2C *repeated start*, and 'P' is I2C *stop*.
- Start each message with 'S', and end each message with 'P'. Each reply will also present this syntax.
- On the host side, send 'S', 'R', and 'P' explicitly; 'N' (*no-acknowledge*) is generated by the translator.
- If either host or target fails to acknowledge, 'N' (followed by 'P') is immediately returned, and the message is terminated.
- Use 'R' when both read and write operations are included in the same command.
- Data is sent and received in hexadecimal characters. To read data, hex characters (typically FF) must be included as placeholders for the reply data. The entire message is echoed back as it actually appeared on I2C, with appropriate data substitutions.
- Because the I2C command interpreter expects all I2C command characters in uppercase, ***QD_ExecuteI2CCommand*** will change them to uppercase in both the command and reply buffers if any are lowercase, but will leave all characters following the terminating 'P' unchanged.

The command string is terminated by the I2C *stop* character 'P' and the response string has CR-LF-null termination represented by '¬'.

Example I2C Command/Response Pairs (spaces shown for clarity)

Target Device	Command	Typical Response	Notes
Digital XD	S 9D FF FF FF FF P	S 9D 01 3A 12 AF P¬	Read pressure
	S 9C R 9F FF FF P	S 9C R 9F FF 3D P¬	Read 2-byte status
	S 9D FF P	S 9D N P¬	Ack Poll (Not Ready)
	S 9D FF P	S 9D 00 P¬	Ack Poll (Ready)
	S AD FF FF FF FF P	S AD 01 02 03 04 P¬	Read current address
	S AC 00 80 R AD FF FF P	S AC 00 80 R AD 01 02 P¬	Read 2 bytes at addr 0080
	S AC 00 84 01 02 03 04 P	S AC 00 84 01 02 03 04 P¬	Write 01020304 to addr 0084
Analog XD comm with on-board Frequency Counter IC	S 99 FF FF FF FF P	S 99 01 3A 12 AF P¬	Read pressure
	S 98 R 9B FF FF P	S 98 R 9B FF 48 P¬	Read 2-byte status
	S 99 FF P	S 99 N P¬	Ack Poll (Not Ready)
	S 99 FF P	S 99 00 P¬	Ack Poll (Ready)
On-Board Memory	S A2 00 00 R A3 FF FF P	S A2 00 00 R A3 0D 01 P¬	Read 2 bytes at addr 0000
	S A2 00 82 03 P	S A2 00 82 03 P¬	Write 03 to addr 0082

SPECIFICATION: QCOM INTERFACE DLL

3.17. QD_ReadUnitADC 0x34

Description	Reads the QCOM microcontroller's 10-bit ADC, which can be selected to read counts representing transducer voltage and transducer current

Prototype	`DWORD QD_ReadUnitADC(`
	`  HANDLE unitHandle,`
	`  BYTE ADCChannel,`
	`  LPDWORD ADCCount);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *ADCChannel*— In: Value that selects the transducer ADC channel to read, as follows: *QD_READ_ADC_CHANNEL_CURRENT* 0x03 *QD_READ_ADC_CHANNEL_VOLTAGE* 0x04 • *ADCCount*— Out: Pointer to the 10-bit ADC count result
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x0016 *QD_ERROR_ADC_COUNT_ZERO* 0x002A *QD_ERROR_NULL_POINTER_PARAMETER* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD currentCount;`
	`DWORD status;`
	`status = QD_ReadUnitADC(`
	`  unitHandle,`
	`  QD_READ_ADC_CHANNEL_CURRENT,`
	`  (LPDWORD) &currentCount);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The transducer current count is %d\n",`
	`  currentCount);`
	`}`
	Output:
	`The transducer current count is 549`

SPECIFICATION: QCOM INTERFACE DLL

3.18. QD_GetTransducerCurrent 0x23

Description	Returns the DC current (in mA) drawn by the transducer

Prototype	`DWORD QD_GetTransducerCurrent(`
	`  HANDLE unitHandle,`
	`  DOUBLE *transducerCurrent);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *transducerCurrent*— Out: Pointer to the transducer current in mA
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING* 0x0026 *QD_ERROR_XD_CURRENT_TOO_HIGH* 0x0027 *QD_ERROR_XD_CURRENT_TOO_LOW*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DOUBLE transducerCurrent;`
	`DWORD status;`
	`status = QD_GetTransducerCurrent(`
	`  unitHandle,`
	`  &transducerCurrent);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The transducer current is %0.2f mA\n",`
	`  transducerCurrent);`
	`}`
	Output:
	`The transducer current is 15.47 mA`

3.19. QD_GetTransducerVoltage 0x25

Description	Returns the DC voltage (in volts) applied to the transducer

Prototype	`DWORD QD_GetTransducerVoltage(`
	`  HANDLE unitHandle,`
	`  DOUBLE *transducerVoltage);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *transducerVoltage*— Out: Pointer to the transducer voltage in volts
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING* 0x0024 *QD_ERROR_XD_VOLTAGE_TOO_HIGH* 0x0025 *QD_ERROR_XD_VOLTAGE_TOO_LOW*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DOUBLE transducerVoltage;`
	`DWORD status;`
	
	`status = QD_GetTransducerVoltage(`
	`  unitHandle,`
	`  &transducerVoltage);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The transducer voltage is %0.2f V\n",`
	`  transducerVoltage);`
	`}`
	
	Output:
	
	`The transducer voltage is 5.12 V`

3.20. QD_SetTransducerPowerState 0x3C

Description	Applies or removes power to the transducer

Prototype	`DWORD QD_SetTransducerPowerState(`
	`  HANDLE unitHandle,`
	`  BYTE powerState);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *powerState*— In: Value used to set the transducer's power state, as follows: *QD_DISABLE_TRANSDUCER_POWER* 0x00 *QD_ENABLE_TRANSDUCER_POWER* 0x01
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	
	`status = QD_SetTransducerPowerState(`
	`  unitHandle,`
	`  QD_ENABLE_TRANSDUCER_POWER);`

3.21. QD_GetModuleControlRegister 0x26

Description	Returns the value of the module's volatile control register

Prototype	`DWORD QD_GetModuleControlRegister(`
	`  HANDLE unitHandle,`
	`  LPBYTE registerValue);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *registerValue*— Out: Pointer to the control register value
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE registerValue;`
	`DWORD status;`
	`status = QD_GetModuleControlRegister(`
	`  unitHandle,`
	`  (LPBYTE) &registerValue);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The control register value is 0x%02X\n",`
	`  registerValue);`
	`}`
	Output:
	`The control register value is 0x02`

The control register bit descriptions are as follows:

Bit #	Description	Default
7	Unassigned	0
6	Unassigned	0
5	Unassigned	0
4	Unassigned	0
3	Unassigned	0
2	Unassigned	0
1	Auto-Load coefficient data when a digital transducer is detected (1 = enabled)	1
0	Host cadence interrupt bit (1 = enabled, 0 = disabled)	0

SPECIFICATION: QCOM INTERFACE DLL

3.22. QD_SetModuleControlRegister 0x3D

Description	Sets the value of the module's volatile control register (see ***QD_GetModuleControlRegister*** for register values)

Prototype	`DWORD QD_SetModuleControlRegister(`
	`  HANDLE unitHandle,`
	`  BYTE registerValue);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *registerValue*— In: Control register value to be written (see ***QD_GetModuleControlRegister*** for possible values)
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	
	`status = QD_SetModuleControlRegister(`
	`  unitHandle,`
	`  0x02);`

3.23. QD_GetModuleStatusRegister 0x2A

Description	Returns the value of module's volatile status register

Prototype	`DWORD QD_GetModuleStatusRegister(`
	`  HANDLE unitHandle,`
	`  LPBYTE registerValue);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *registerValue*— Out: Pointer to the status register value
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE registerValue = 0;`
	`DWORD status;`
	`status = QD_GetModuleStatusRegister(`
	`  unitHandle,`
	`  (LPBYTE) &registerValue);`
	`if (status == QD_SUCCESS)`
	`  printf("The status register value is 0x%02X\n",`
	`  registerValue);`
	Output:
	`The status register value is 0x2C`

The status register bit descriptions are as follows:

Bit	Description	Value	Meaning
7	Error Present	1	Internal error present
		0	No internal error present
6	High-Side Power Switch OC-TSD	1	High side switch in over-current and/or thermal shut down condition
		0	High side switch not in over-current and/or thermal shut down condition
5	Transducer Power Status	1	Transducer power control signal on
		0	Transducer power control signal off
4	Transducer Over-Current	1	Transducer is in over-current condition
		0	Transducer is not in over-current condition
3	Transducer Present	1	Transducer is present (transducer current draw > 2 mA)
		0	Transducer is not present or is not drawing current
2	Transducer Type (bit2)	`000b`	Transducer is not present and/or is not detected
1	Transducer Type (bit1)	`001b`	Analog/Frequency transducer present
0	Transducer Type (bit0)	`010b`	Pre-FireFly digital transducer present, with memory
		`011b`	FireFly digital transducer present (with V_{REF})
		`100b`	FireFly digital transducer present (without V_{REF})
		`101b`	Digital transducer present with no memory

3.24. QD_GetModuleFirmwareID 0x27

Description	Returns the module's firmware ID and version

Prototype	`DWORD QD_GetModuleFirmwareID(`
	`  HANDLE unitHandle,`
	`  LPBYTE firmwareID);`

Parameters	• *unitHandle*—In: Module handle returned by ***QD_Open*** • *firmwareID*— Out: Array of 4 bytes that represent the firmware ID and version of the target module, as follows: `Byte 0` Company ID (Quartzdyne = 0x0D) `Byte 1` Product ID (firmware = 0x16) `Byte 2` Firmware major version (1 or A = 0x01) `Byte 3` Firmware minor version (2 = 0x02)
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE  firmwareID[QD_FIRMWARE_ID_LENGTH];`
	`DWORD  status;`
	
	`status = QD_GetModuleFirmwareID(`
	`  unitHandle,`
	`  (LPBYTE) firmwareID);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The module firmware ID is %02X %02X %02X %02X\n",`
	`  firmwareID[0], firmwareID[1], firmwareID[2], firmwareID[3]);`
	`}`
	
	Output:
	
	`The module firmware ID is 0D 16 01 02`

SPECIFICATION: QCOM INTERFACE DLL

3.25. QD_SetCadenceTimer 0x36

Description	Sets the free-running, "heartbeat" cadence timer used to acquire and refresh the transducer counts

Prototype	`DWORD QD_SetCadenceTimer(`
	`  HANDLE unitHandle,`
	`  DWORD cadenceTimerValue);`

Parameters	• *unitHandle*—In: Module handle returned by ***QD_Open*** • *cadenceTimerValue*— In: Cadence timer value in ms to be written. Usable range is 250 to 2000, inclusive (default = 2000 ms). If a value is set outside this range, it will be adjusted to an appropriate value.
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	
	`status = QD_SetCadenceTimer(`
	`  unitHandle,`
	`  350);`

SPECIFICATION: QCOM INTERFACE DLL

3.26. QD_GetCadenceTimer 0x18

Description	Returns the current value of the cadence timer described in ***QD_SetCadenceTimer***

Prototype	`DWORD QD_GetCadenceTimer(`
	`  HANDLE unitHandle,`
	`  LPDWORD cadenceTimerValue);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *cadenceTimerValue*— Out: Pointer to the cadence timer value in ms (250 through 2000, inclusive)
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD timerValue;`
	`DWORD status;`
	
	`status = QD_GetCadenceTimer(`
	`  unitHandle,`
	`  (LPDWORD) &timerValue);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The cadence timer value is %d ms\n",`
	`  timerValue);`
	`}`
	
	Output:
	
	`The cadence timer value is 350 ms`

SPECIFICATION: QCOM INTERFACE DLL

3.27. QD_GetModuleModeSwitchSetting 0x28

Description	Returns the module's mode switch setting during the development phase of the project (and/or allow reading of the mode switch in future designs as needed)

Prototype	`DWORD QD_GetModuleModeSwitchSetting(`
	`  HANDLE unitHandle,`
	`  LPBYTE modeSwitchSetting);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *modeSwitchSetting*— Out: Pointer to the mode switch setting, as follows: *QD_MODE_SWITCH_PRESSED* 0x00 *QD_MODE_SWITCH_NOT_PRESSED* 0x01
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE switchSetting;`
	`DWORD status;`
	
	`status = QD_GetModuleModeSwitchSetting(`
	`  unitHandle,`
	`  (LPBYTE) &switchSetting);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The mode switch is %sdepressed\n",`
	`  (switchSetting ? "not " : ""));`
	`}`
	
	Output:
	
	`The mode switch is not depressed`

3.28. QD_SetFirmwareMode 0x38

Description	Transitions the module firmware to Boot Loader Mode or Application Mode. Boot Loader Mode is used for firmware programming, and Application Mode is the normal operating mode of the firmware.

Prototype	`DWORD QD_SetFirmwareMode(`
	`  HANDLE unitHandle,`
	`  BYTE mode);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *mode*— In: Firmware mode of the module as follows: *QD_SET_APPLICATION_MODE* 0x00 *QD_SET_BOOT_LOADER_MODE* 0x01
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	
	`status = QD_SetFirmwareMode(`
	`  unitHandle,`
	`  QD_SET_BOOT_LOADER_MODE);`

SPECIFICATION: QCOM INTERFACE DLL

3.29. QD_ResetModule 0x35

Description	Cycles the target module's FPGA / ASIC (counter) reset line with a 2 ms strobe
Prototype	`DWORD QD_ResetModule(` `  HANDLE unitHandle);`
Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open***
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	
	`status = QD_ResetModule(`
	`  unitHandle);`

3.30. QD_GetInternalErrorCode 0x1C

Description	Retrieves the value of the target module's internal error code register

Prototype	`DWORD QD_GetInternalErrorCode(`
	`  HANDLE unitHandle,`
	`  LPWORD errorCode);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *errorCode*— Out: Pointer to the module's internal error code, as follows: `0x0000` No internal error detected `0x0001` Over-current error (the transducer is drawing too much current)
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`WORD errorCode;`
	`DWORD status;`
	
	`status = QD_GetInternalErrorCode(`
	`  unitHandle,`
	`  (LPWORD) &errorCode);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The internal error code is 0x%04X\n",`
	`  errorCode);`
	`  status = QD_ClearInternalrror(unitHandle);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  status = QD_GetInternalErrorCode(`
	`  unitHandle,`
	`  (LPWORD) &errorCode);`
	`  printf("The internal error code is now 0x%04X\n",`
	`  errorCode);`
	`  }`
	`}`
	
	Output:
	
	`The internal error code is 0x0001`
	`The internal error code is now 0x0000`

SPECIFICATION: QCOM INTERFACE DLL

3.31. QD_ClearInternalError 0x13

Description	Clears the target module's internal error code register. It is recommended that an application call ***QD_GetInternalErrorCode*** following a call to ***QD_ClearInternalError*** to verify the internal error code was actually cleared (and also to verify that whatever caused the error initially does not recur).
Prototype	`DWORD QD_ClearInternalError(` `  HANDLE unitHandle);`
Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open***
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`WORD errorCode;`
	`DWORD status;`
	
	`status = QD_GetInternalErrorCode(`
	`  unitHandle,`
	`  (LPWORD) &errorCode);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The internal error code is 0x%04X\n",`
	`  errorCode);`
	`  status = QD_ClearInternalrror(unitHandle);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  status = QD_GetInternalErrorCode(`
	`  unitHandle,`
	`  (LPWORD) &errorCode);`
	`  printf("The internal error code is now 0x%04X\n",`
	`  errorCode);`
	`  }`
	`}`
	
	Output:
	`The internal error code is 0x0001`
	`The internal error code is now 0x0000`

3.32. QD_SetI2CDataRate 0x3A

Description	Sets the module's I2C data rate to communicate with the transducer

Prototype	`DWORD QD_SetI2CDataRate(`
	`  HANDLE unitHandle,`
	`  DOUBLE newDataRate);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *newDataRate*— In: The I2C data rate in kHz to set the module. The usable range is 33.3 to 100.0 (default = 33.3), inclusive. If a value is written outside of this range, it will be adjusted to an appropriate value.
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	
	`status = QD_SetI2CDataRate(`
	`  unitHandle,`
	`  76.9);`

SPECIFICATION: QCOM INTERFACE DLL

3.33. QD_GetI2CDataRate 0x1B

Description	Retrieves the module's current I2C data rate setting to communicate with the transducer

Prototype	`DWORD QD_GetI2CDataRate(`
	`  HANDLE unitHandle,`
	`  DOUBLE *currentDataRate);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *currentDataRate*— Out: Pointer to the module's current I2C data rate setting in kHz, whose value is described in ***QD_SetI2CDataRate***
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DOUBLE dataRate;`
	`DWORD status;`
	
	`status = QD_GetI2CDataRate(`
	`  unitHandle,`
	`  &dataRate);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The current I2C data rate is %0.1f kHz\n",`
	`  dataRate);`
	`}`
	
	Output:
	
	`The current I2C data rate is 76.9 kHz`

3.34. QD_CalculateFirmwarePageCRC N/A

Description	Calculates the CRC of the specified module firmware page of the specified firmware data in the buffer

Prototype	`DWORD QD_CalculateFirmwarePageCRC(`
	`  LPBYTE firmwareData,`
	`  BYTE pageNumber);`

Parameters	• *firmwareData*— In: 64512 (63 X 1024)-byte buffer of valid firmware data • *pageNumber*— In: Firmware page whose CRC is to be calculated
Return Value	*The 32-bit CRC value of the specified firmware page data*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE  firmwareData[QD_FIRMWARE_MAXIMUM_DATA_SIZE];`
	`DWORD  firmwareCRC;`
	`DWORD  status;`
	
	`status = QD_ReadFirmwareDataFromFile(`
	`  (LPBYTE) "FW0752.hex",`
	`  (LPBYTE) firmwareData);`
	`if (status == QD_SUCCESS)`
	`{`
	`  DWORD  firmwareCRC;`
	
	`  firmwareCRC = QD_CalculateFirmwarePageCRC(`
	`  (LPBYTE) firmwareData,`
	`  0x01);`
	`  printf("The CRC of firmware page 1 is 0x%08X\n",`
	`  firmwareCRC);`
	`}`
	
	Output:
	
	`The CRC of firmware page 1 is 0x000085FE`

SPECIFICATION: QCOM INTERFACE DLL

3.35. QD_CalculatePressureAndTemperature 0x10

Description	Calculates and returns both the pressure value in PSI and temperature value in Celsius, given the pressure and temperature counts, along with the coefficient data

Prototype	`DWORD QD_CalculatePressureAndTemperature(`
	`  LPBYTE coefficientData,`
	`  DWORD pressureCount,`
	`  DWORD temperatureCount,`
	`  DOUBLE *pressurePSI,`
	`  DOUBLE *temperatureCelsius);`

Parameters	• *coefficientData*— In: 256-byte buffer of coefficient file data • *pressureCount*— In: The pressure count value retrieved from the transducer • *temperatureCount*— In: The temperature count value retrieved from the transducer • *pressurePSI*— Out: Pointer to the calculated pressure value in PSI • *temperatureCelsius*— Out: Pointer to the calculated temperature value in Celsius
Return Value	`0x0000` *QD_SUCCESS*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD pressureCount;`
	`DWORD temperatureCount;`
	`DWORD status;`
	
	`status = QD_GetTransducerCounts(`
	`  unitHandle,`
	`  (LPDWORD) &pressureCount,`
	`  (LPDWORD) &temperatureCount);`
	`if (status == QD_SUCCESS)`
	`{`
	`  BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`  DOUBLE pressure;`
	`  DOUBLE temperature;`
	
	`  QD_ReadCoefficientDataFromDevice(`
	`  unitHandle,`
	`  (LPBYTE) coefficientData);`
	`  status = QD_CalculatePressureAndTemperature(`
	`  (LPBYTE) coefficientData,`
	`  pressureCount,`
	`  temperatureCount,`
	`  &pressure,`
	`  &temperature);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  printf("The pressure is %0.2f PSI and the temperature is %0.2f °C\n",`
	`  pressure, temperature);`
	`  }`
	`}`
	
	Output:
	
	`The pressure is 13.78 PSI and the temperature is 24.04 °C`

3.36. QD_CoefficientDataIsValid N/A

Description	Determines whether the specified buffer contains valid coefficient data
Prototype	`bool QD_CoefficientDataIsValid(` `  LPBYTE coefficientData);`
Parameters	• *coefficientData*— In: 256-byte buffer of supposed coefficient file data
Return Value	`true` *The data appears to be valid coefficient data* `false` *The data does not meet the criteria for valid coefficient data*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`bool valid;`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	`status = QD_ReadCoefficientDataFromDevice(`
	`  unitHandle,`
	`  (LPBYTE) coefficientData);`
	`if (status == QD_SUCCESS)`
	`{`
	`  valid = QD_CoefficientDataIsValid(`
	`  (LPBYTE) coefficientData);`
	`  printf("The coefficient data %s valid\n",`
	`  (valid ? "is" : "is not"));`
	`}`
	Output:
	`The coefficient data is valid`

SPECIFICATION: QCOM INTERFACE DLL

3.37. QD_CoefficientHexFileDataIsValid 0x12

Description	Determines whether the specified buffer contains valid coefficient hex file data
Prototype	`bool QD_CoefficientHexFileDataIsValid(` `  LPBYTE hexFileData);`
Parameters	• *hexFileData*— In: Buffer of supposed coefficient hex file data
Return Value	`true` *The data appears to be valid coefficient hex file data* `false` *The data does not meet the criteria for valid coefficient hex file data*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`bool valid;`
	`long bytesRead;`
	`long fileSize;`
	`errno_t result;`
	`BYTE *hexFileData;`
	`FILE *filePtr;`
	`result = fopen_s(&filePtr, "XD242109.hex", "rb");`
	`if (filePtr && (result == 0))`
	`{`
	`  fseek(filePtr, 0, SEEK_END);`
	`  fileSize = ftell(filePtr);`
	`  fseek(filePtr, 0, SEEK_SET);`
	`  hexFileData = (LPBYTE) malloc(fileSize);`
	`  if (hexFileData)`
	`  {`
	`  bytesRead = fread((void *) hexFileData, 1, fileSize, filePtr);`
	`  if (bytesRead == fileSize)`
	`  {`
	`  valid = QD_CoefficientHexFileDataIsValid(`
	`  (LPBYTE) hexFileData);`
	`  printf("The coefficient hex file data %s valid\n",`
	`  (valid ? "is" : "is not"));`
	`  }`
	`  free((void *) hexFileData);`
	`  }`
	`  fclose(filePtr);`
	`}`
	Output:
	`The coefficient hex file data is valid`

3.38. QD_DataIsInHexFileFormat N/A

Description	Determines whether the specified buffer contains data that is structured in the Intel Hex File Format **Note: *QD_DataIsInHexFileFormat* determines whether the data appears to be formatted as an Intel Hex File, while *QD_HexFileFormatIsValid* determines whether the data conforms strictly to the Intel Hex File Format.**
Prototype	`bool QD_DataIsInHexFileFormat(` `  LPBYTE hexFileData);`
Parameters	• *hexFileData*— In: Buffer of supposed hex file data
Return Value	`true` *The data appears to be valid hex file data* `false` *The data does not meet the criteria for valid hex file data*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`bool valid;`
	`long bytesRead;`
	`long fileSize;`
	`errno_t result;`
	`BYTE *hexFileData;`
	`FILE *filePtr;`
	
	`result = fopen_s(&filePtr, "XD242109.hex", "rb");`
	`if (filePtr && (result == 0))`
	`{`
	`  fseek(filePtr, 0, SEEK_END);`
	`  fileSize = ftell(filePtr);`
	`  fseek(filePtr, 0, SEEK_SET);`
	`  hexFileData = (LPBYTE) malloc(fileSize);`
	`  if (hexFileData)`
	`  {`
	`  bytesRead = fread((void *) hexFileData, 1, fileSize, filePtr);`
	`  if (bytesRead == fileSize)`
	`  {`
	`  valid = QD_DataIsInHexFileFormat(`
	`  (LPBYTE) hexFileData);`
	`  printf("The data %s in hex file format\n",`
	`  (valid ? "is" : "is not"));`
	`  }`
	`  free((void *) hexFileData);`
	`  }`
	`  fclose(filePtr);`
	`}`
	
	Output:
	
	`The data is in hex file format`

3.39. QD_EraseFirmwarePage 0x15

Description	Erases the firmware data from the specified module firmware page (in particular, used to erase the last firmware page just prior to writing new data to any of the firmware pages) **Note: This function must only be called in Boot Loader Mode**

Prototype	`DWORD QD_EraseFirmwarePage(`
	`  HANDLE unitHandle,`
	`  BYTE pageNumber);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *pageNumber*— In: Firmware page to be erased
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE lastPage = QD_FIRMWARE_LAST_PAGE_C8051F320;`
	`BYTE pageNumber;`
	`BYTE firmwareData[QD_FIRMWARE_MAXIMUM_DATA_SIZE];`
	`WORD pageCRC;`
	`DWORD status;`
	
	`QD_ReadFirmwareDataFromFile(`
	`  (LPBYTE) "FW0752.hex",`
	`  (LPBYTE) firmwareData);`
	`QD_SetFirmwareKeyCodes(unitHandle, QD_FIRMWARE_KEY_0, QD_FIRMWARE_KEY_1);`
	`status = EraseFirmwarePage(`
	`  unitHandle,`
	`  lastPage);`
	`if (status == QD_SUCCESS)`
	`{`
	`  for (pageNumber = QD_FIRMWARE_FIRST_PAGE_C8051F320;`
	`  ((pageNumber <= lastPage) && (status == QD_SUCCESS)); pageNumber++)`
	`  {`
	`  pageCRC = 0;`
	`  status = QD_WriteFirmwarePage(`
	`  unitHandle,`
	`  (LPBYTE) firmwareData,`
	`  pageNumber,`
	`  (LPWORD) &pageCRC);`
	`  }`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  QD_WriteFirmwareSignature(unitHandle);`
	`  QD_SetFirmwareKeyCodes(unitHandle, 0, 0);`
	`  }`
	`}`

3.40. QD_FirmwareHexFileDataIsValid N/A

Description	Determines whether the specified buffer contains valid firmware hex file data
Prototype	`bool QD_FirmwareHexFileDataIsValid(` `  LPBYTE hexFileData);`
Parameters	• *hexFileData*— In: Buffer of supposed firmware hex file data
Return Value	`true` *The data appears to be valid firmware hex file data* `false` *The data does not meet the criteria for valid firmware hex file data*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`bool valid;`
	`long bytesRead;`
	`long fileSize;`
	`errno_t result;`
	`BYTE *hexFileData;`
	`FILE *filePtr;`
	`result = fopen_s(&filePtr, "FW0752.hex", "rb");`
	`if (filePtr && (result == 0))`
	`{`
	`  fseek(filePtr, 0, SEEK_END);`
	`  fileSize = ftell(filePtr);`
	`  fseek(filePtr, 0, SEEK_SET);`
	`  hexFileData = (LPBYTE) malloc(fileSize);`
	`  if (hexFileData)`
	`  {`
	`  bytesRead = fread((void *) hexFileData, 1, fileSize, filePtr);`
	`  if (bytesRead == fileSize)`
	`  {`
	`  valid = QD_FirmwareHexFileDataIsValid(`
	`  (LPBYTE) hexFileData);`
	`  printf("The firmware hex file data %s valid\n",`
	`  (valid ? "is" : "is not"));`
	`  }`
	`  free((void *) hexFileData);`
	`  }`
	`  fclose(filePtr);`
	`}`
	Output:
	`The firmware hex file data is valid`

3.41. QD_FormatCoefficientHexFileData N/A

Description	Formats the specified unformatted data into a hex file data format, specifically constructed for coefficient data

Prototype	`DWORD QD_FormatCoefficientHexFileData(`
	`  LPBYTE unformattedData,`
	`  LPBYTE formattedData);`

Parameters	• *unformattedData*— In: Null-terminated buffer of raw coefficient data • *formattedData*— Out: 734-byte buffer to store the formatted coefficient hex file data
Return Value	*The number of bytes actually stored in the formatted buffer, not including the null termination*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`char *filename = "CF_New.hex";`
	`long bytesRead;`
	`long fileSize;`
	`errno_t result;`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`BYTE hexFileData[QD_COEFFICIENT_DATA_HEX_FILE_SIZE + 1];`
	`FILE *filePtr;`
	`DWORD status;`
	`status = QD_ReadCoefficientDataFromDevice(`
	`unitHandle,`
	`(LPBYTE) coefficientData);`
	`if (status == QD_SUCCESS)`
	`{`
	`fileSize = QD_FormatCoefficientHexFileData(`
	`(LPBYTE) coefficientData,`
	`(LPBYTE) hexFileData);`
	`result = fopen_s(&filePtr, fileName, "wb");`
	`if (filePtr && (result == 0))`
	`{`
	`bytesRead = fwrite((void *) hexFileData, 1, fileSize, filePtr);`
	`if (bytesRead == fileSize)`
	`{`
	`printf("The coefficient hex file %s was successfully created\n",`
	`fileName);`
	`}`
	`fclose(filePtr);`
	`}`
	`}`
	Output:
	`The coefficient hex file CF_New.hex was successfully created`

3.42. QD_GetFirmwareCRC 0x19

Description	Retrieves the CRC value for the specified firmware page in the firmware installed in the unit **Note: This function must only be called in Boot Loader Mode**

Prototype	`DWORD QD_GetFirmwareCRC(`
	`  HANDLE unitHandle,`
	`  BYTE pageNumber,`
	`  LPWORD pageCRC);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *pageNumber*— In: Firmware page whose CRC to calculate • *pageCRC*— Out: The computed CRC value for the specified firmware page
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	`WORD firmwareCRC;`
	
	`status = QD_GetFirmwareCRC(`
	`  unitHandle,`
	`  0x01,`
	`  (LPWORD) &firmwareCRC);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The CRC of firmware page 1 is 0x%04X\n",`
	`  firmwareCRC);`
	`}`
	
	Output:
	
	`The CRC of firmware page 1 is 0x85FE`

SPECIFICATION: QCOM INTERFACE DLL

3.43. QD_GetFirmwareDeviceInfo 0x1A

Description	Retrieves the structure of firmware device information for the module **Note: This function must only be called in Boot Loader Mode**

Prototype	`DWORD QD_GetFirmwareDeviceInfo(`
	`  HANDLE unitHandle,`
	`  LPBYTE unitInfo);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open***
	• *unitInfo*— Out: Pointer to a 9-byte *UnitDeviceInfoDef* structure to store the device information, as follows:
	
	`BYTE bootloaderFWVersionHigh;`
	`BYTE bootloaderFWVersionLow;`
	`BYTE libraryFWVersionHigh;`
	`BYTE libraryFWVersionLow;`
	`BYTE applicationFWVersionHigh;`
	`BYTE applicationFWVersionLow;`
	`WORD signature;`
	`BYTE deviceCode;`
	
	The descriptions for the applicable fields are as follows:
	
	• *applicationFWVersionHigh*— Firmware major version
	• *applicationFWVersionLow*— Firmware minor version
	• *signature*— Firmware code signature (should be 0x3DC2)
	• *deviceCode*— Target device code (device for which the firmware was designed; should be 0x81)

Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	`UnitDeviceInfoDef moduleInfo;`
	
	`status = QD_GetFirmwareDeviceInfo(`
	`  unitHandle,`
	`  (LPBYTE) &moduleInfo);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The installed firmware version is %d.%d\n",`
	`  moduleInfo.applicationFWVersionHigh,`
	`  moduleInfo.applicationFWVersionLow);`
	`}`
	
	Output:
	
	`The installed firmware version is 1.2`

3.44. QD_GetUSBDLLVersion 0x2B

Description	Retrieves the version of the installed Silicon Labs USB DLL

Prototype	`DWORD QD_GetUSBDLLVersion(`
	`  LPDWORD upperVersion,`
	`  LPDWORD lowerVersion);`

Parameters	• *upperVersion*— Out: Pointer to the higher two words of the DLL version • *lowerVersion*— Out: Pointer to the lower two words of the DLL version
Return Value	`0x0000` *QD_SUCCESS* `Nonzero` *The Silicon Labs DLL is not installed or could not be found*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD lowerVersion;`
	`DWORD upperVersion;`
	`DWORD status;`
	
	`status = QD_GetUSBDLLVersion(`
	`  (LPDWORD) &upperVersion,`
	`  (LPDWORD) &lowerVersion);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The Silicon Labs DLL version is %X.%X.%X.%X\n",`
	`  ((upperVersion >> 16) & 0xFFFF), (upperVersion & 0xFFFF),`
	`  ((lowerVersion >> 16) & 0xFFFF), (lowerVersion & 0xFFFF));`
	`}`
	
	Output:
	
	`The Silicon Labs DLL version is 3.3.0.0`

SPECIFICATION: QCOM INTERFACE DLL

3.45. QD_GetUSBDriverVersion 0x2C

Description	Retrieves the version of the installed Silicon Labs USB driver

Prototype	`DWORD QD_GetUSBDriverVersion(`
	`  LPDWORD upperVersion,`
	`  LPDWORD lowerVersion);`

Parameters	• *upperVersion*— Out: Pointer to the higher two words of the driver version • *lowerVersion*— Out: Pointer to the lower two words of the driver version
Return Value	`0x0000` *QD_SUCCESS* `Nonzero` *The Silicon Labs driver is not installed or could not be found (Note: This function might fail on Windows Server 2003)*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD lowerVersion;`
	`DWORD upperVersion;`
	`DWORD status;`
	
	`status = QD_GetUSBDriverVersion(`
	`  (LPDWORD) &upperVersion,`
	`  (LPDWORD) &lowerVersion);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The Silicon Labs driver version is %X.%X.%X.%X\n",`
	`  ((upperVersion >> 16) & 0xFFFF), (upperVersion & 0xFFFF),`
	`  ((lowerVersion >> 16) & 0xFFFF), (lowerVersion & 0xFFFF));`
	`}`
	
	Output:
	
	`The Silicon Labs driver version is 3.3.0.0`

SPECIFICATION: QCOM INTERFACE DLL

3.46. QD_GetModuleSerialNumber 0x29

Description	Retrieves the module serial number string

Prototype	`DWORD QD_GetModuleSerialNumber(`
	`  DWORD unitNumber,`
	`  LPBYTE serialNumber);`

Parameters	• *unitNumber*— In: Index of the module for which the product description string or serial number string is requested • *serialNumber*— Out: 256-byte buffer to store the serial number string (Note: The string will be in all uppercase.)
Return Value	`0x0000` *QD_SUCCESS* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x00FF` *QD_ERROR_NO_DEVICE_FOUND*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	`DWORD unitNumber = 2;`
	`BYTE serialNumber[QD_MAXIMUM_SERIAL_NUMBER_SIZE];`
	
	`status = QD_GetoduleSerialNumber(`
	`  unitNumber,`
	`  (LPBYTE) serialNumber);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The serial number of module %d is %s\n",`
	`  unitNumber,`
	`  serialNumber);`
	`}`
	
	Output:
	
	`The serial number of module 2 is Q0529`

3.47. QD_HexFileFormatIsValid N/A

Description	Determines whether the specified buffer contains data that conforms to the Intel Hex File Format

Prototype	`bool QD_HexFileFormatIsValid(`
	`  LPBYTE hexFileData);`
	
	Note: *QD_DataIsInHexFileFormat* determines whether the data appears to be formatted as an Intel Hex File, while *QD_HexFileFormatIsValid* determines whether the data conforms strictly to the Intel Hex File Format.

Parameters	• *hexFileData*— In: Buffer of supposed hex file data
Return Value	`true` *The data appears to be valid hex file data* `false` *The data does not meet the criteria for valid hex file data*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`bool valid;`
	`long bytesRead;`
	`long fileSize;`
	`errno_t result;`
	`BYTE *hexFileData;`
	`FILE *filePtr;`
	
	`result = fopen_s(&filePtr, "XD242109.hex", "rb");`
	`if (filePtr && (result == 0))`
	`{`
	`  fseek(filePtr, 0, SEEK_END);`
	`  fileSize = ftell(filePtr);`
	`  fseek(filePtr, 0, SEEK_SET);`
	`  hexFileData = (LPBYTE) malloc(fileSize);`
	`  if (hexFileData)`
	`  {`
	`  bytesRead = fread((void *) hexFileData, 1, fileSize, filePtr);`
	`  if (bytesRead == fileSize)`
	`  {`
	`  valid = QD_HexFileFormatIsValid(`
	`  (LPBYTE) hexFileData);`
	`  printf("The hex file format %s valid\n",`
	`  (valid ? "is" : "is not"));`
	`  }`
	`  free((void *) hexFileData);`
	`  }`
	`  fclose(filePtr);`
	`}`
	
	Output:
	
	`The hex file format is valid`

SPECIFICATION: QCOM INTERFACE DLL

3.48. QD_Read 0x2E

Description	Reads data from the specified module

Prototype	`DWORD QD_Read(`
	`  HANDLE unitHandle,`
	`  LPBYTE readBuffer,`
	`  DWORD readBufferSize,`
	`  LPDWORD bytesRead);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *readBuffer*— Out: Buffer into which read bytes will be copied • *readBufferSize*— In: Size in bytes of the read buffer • *bytesRead*— Out: Pointer to the number of bytes actually read
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE command[QD_COMMAND_BUFFER_SIZE];`
	`DWORD bytesTransmitted;`
	`DWORD commandLength;`
	`DWORD status;`
	
	`command[0] = 0x08;`
	`commandLength = 1;`
	`status = QD_Write(`
	`  unitHandle,`
	`  (LPBYTE) command,`
	`  commandLength,`
	`  (LPDWORD) &bytesTransmitted);`
	`if ((status == QD_SUCCESS) && (bytesTransmitted == commandLength))`
	`{`
	`  commandLength = QD_FIRMWARE_ID_LENGTH;`
	`  status = QD_WaitForReply(unitHandle, commandLength);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  status = QD_Read(`
	`  unitHandle,`
	`  (LPBYTE) command,`
	`  commandLength,`
	`  (LPDWORD) &bytesTransmitted);`
	`  if ((status == QD_SUCCESS) && (bytesTransmitted == commandLength))`
	`  printf("The firmware ID is %02X %02X %02X %02X\n",`
	`  command[0], command[1], command[2], command[3]);`
	`  }`
	`}`
	
	Output:
	`The firmware ID is 0D 16 01 02`

SPECIFICATION: QCOM INTERFACE DLL

3.49. QD_ReadCoefficientDataFromDevice 0x2F

Description	Retrieves the coefficient data from transducer memory if a digital transducer with memory is attached, or from QCOM memory otherwise

Prototype	`DWORD QD_ReadCoefficientDataFromDevice(`
	`  HANDLE unitHandle,`
	`  LPBYTE coefficientData);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *coefficientData*— Out: 256-byte buffer to store the coefficient file data
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING* ‡ See the exception note for ***Next Byte*** in section 4

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	
	`status = QD_ReadCoefficientDataFromDevice(`
	`  unitHandle,`
	`  (LPBYTE) coefficientData);`

SPECIFICATION: QCOM INTERFACE DLL

3.50. QD_ReadCoefficientDataFromModulePage 0x32

Description	Retrieves the coefficient data from the module at the specified page

Prototype	`DWORD QD_ReadCoefficientDataFromModulePage(`
	`  HANDLE unitHandle,`
	`  BYTE pageNumber,`
	`  LPBYTE coefficientData);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *pageNumber*— In: Memory page from which to read the data, as follows: `0x00` Page at offset 0x0000 `0x01` Page at offset 0x0100 `0x02` Page at offset 0x0200 `0x03` Page at offset 0x0300 • *coefficientData*— Out: 256-byte buffer to store the coefficient file data
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING* ‡ See the exception note for ***Next Byte*** in section 4

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	`status = QD_ReadCoefficientDataFromModulePage(`
	`  unitHandle,`
	`  0,`
	`  (LPBYTE) coefficientData);`

SPECIFICATION: QCOM INTERFACE DLL

3.51. QD_ReadFirmwareDataFromFile 0x33

Description	Retrieves the firmware data from the specified file path

Prototype	`DWORD QD_ReadFirmwareDataFromFile(`
	`  LPBYTE pathName,`
	`  LPBYTE firmwareData);`

Parameters	• *pathName*— In: Null-terminated string that includes the file name and the path name of the firmware hex data file • *firmwareData*— Out: 64512 (63 X 1024)-byte buffer to store the firmware data
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING* 0x0014 *QD_ERROR_MEMORY_ALLOCATION_FAILED* 0x0015 *QD_ERROR_FILE_OPEN_FAILURE* 0x0017 *QD_ERROR_FILE_SIZE_MISMATCH* 0x0018 *QD_ERROR_INVALID_HEX_FILE_DATA* 0x002A *QD_ERROR_NULL_POINTER_PARAMETER* 0x002B *QD_ERROR_ZERO_LENGTH_STRING_PARAMETER* 0x0070 *QD_ERROR_FILE_NOT_FOUND*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE firmwareData[QD_FIRMWARE_MAXIMUM_DATA_SIZE];`
	`DWORD status;`
	
	`status = QD_ReadFirmwareDataFromFile(`
	`  (LPBYTE) "FW0752.hex",`
	`  (LPBYTE) firmwareData);`

SPECIFICATION: QCOM INTERFACE DLL

3.52. QD_SetFirmwareKeyCodes 0x37

Description	Sets the flash key codes in the firmware flash memory **Note: This function must only be called in Boot Loader Mode**

Prototype	`DWORD QD_SetFirmwareKeyCodes(`
	`  HANDLE unitHandle,`
	`  BYTE key0,`
	`  BYTE key1);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *key0*— In: First key byte (QD_FIRMWARE_KEY_0) • *key1*— In: Second key byte (QD_FIRMWARE_KEY_1)
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE lastPage = QD_FIRMWARE_LAST_PAGE_C8051F320;`
	`BYTE pageNumber;`
	`BYTE firmwareData[QD_FIRMWARE_MAXIMUM_DATA_SIZE];`
	`WORD pageCRC;`
	`DWORD status;`
	
	`QD_ReadFirmwareDataFromFile(`
	`  (LPBYTE) "FW0752.hex",`
	`  (LPBYTE) firmwareData);`
	`QD_SetFirmwareKeyCodes(`
	`  unitHandle,`
	`  QD_FIRMWARE_KEY_0,`
	`  QD_FIRMWARE_KEY_1);`
	`status = EraseFirmwarePage(unitHandle, lastPage);`
	`if (status == QD_SUCCESS)`
	`{`
	`  for (pageNumber = QD_FIRMWARE_FIRST_PAGE_C8051F320;`
	`  ((pageNumber <= lastPage) && (status == QD_SUCCESS)); pageNumber++)`
	`  {`
	`  pageCRC = 0;`
	`  status = QD_WriteFirmwarePage(`
	`  unitHandle,`
	`  (LPBYTE) firmwareData,`
	`  pageNumber,`
	`  (LPWORD) &pageCRC);`
	`  }`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  QD_WriteFirmwareSignature(unitHandle);`
	`  QD_SetFirmwareKeyCodes(unitHandle, 0, 0);`
	`  }`
	`}`

SPECIFICATION: QCOM INTERFACE DLL

3.53. QD_SetFirmwarePage 0x39

Description	Sets the firmware page on which to perform subsequent firmware operations **Note: This function must only be called in Boot Loader Mode**

Prototype	`DWORD QD_SetFirmwarePage(`
	`  HANDLE unitHandle,`
	`  BYTE pageNumber);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *pageNumber*— In: Firmware page number to set
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE lastPage = QD_FIRMWARE_LAST_PAGE_C8051F320;`
	`BYTE pageNumber;`
	`BYTE firmwareData[QD_FIRMWARE_MAXIMUM_DATA_SIZE];`
	`WORD pageCRC;`
	`DWORD status;`
	
	`QD_ReadFirmwareDataFromFile(`
	`  (LPBYTE) "FW0752.hex",`
	`  (LPBYTE) firmwareData);`
	`QD_SetFirmwareKeyCodes(unitHandle, QD_FIRMWARE_KEY_0, QD_FIRMWARE_KEY_1);`
	`status = QD_SetFirmwarePage(`
	`  unitHandle,`
	`  lastPage);`
	`status = EraseFirmwarePage(unitHandle, lastPage);`
	`if (status == QD_SUCCESS)`
	`{`
	`  for (pageNumber = QD_FIRMWARE_FIRST_PAGE_C8051F320;`
	`  ((pageNumber <= lastPage) && (status == QD_SUCCESS)); pageNumber++)`
	`  {`
	`  pageCRC = 0;`
	`  QD_SetFirmwarePage(unitHandle, pageNumber);`
	`  status = QD_WriteFirmwarePage(`
	`  unitHandle,`
	`  (LPBYTE) firmwareData,`
	`  pageNumber,`
	`  (LPWORD) &pageCRC);`
	`  }`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  QD_WriteFirmwareSignature(unitHandle);`
	`  QD_SetFirmwareKeyCodes(unitHandle, 0, 0);`
	`  }`
	`}`

SPECIFICATION: QCOM INTERFACE DLL

3.54. QD_UnFormatHexFileData N/A

Description	Converts the specified hex file data into its memory-mapped form

Prototype	`DWORD QD_UnFormatHexFileData(`
	`  LPBYTE formattedData,`
	`  LPBYTE unformattedData);`

Parameters	• *formattedData*— In: Buffer of hex file data • *unformattedData*— Out: Buffer to store the raw data with hex file formatting removed
Return Value	The number of unformatted bytes

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`long bytesRead;`
	`long fileSize;`
	`DWORD dataSize;`
	`errno_t result;`
	`BYTE *hexFileData;`
	`BYTE *dataBuffer;`
	`FILE *filePtr;`
	
	`result = fopen_s(&filePtr, "XD242109.hex", "rb");`
	`if (filePtr && (result == 0))`
	`{`
	`  fseek(filePtr, 0, SEEK_END);`
	`  fileSize = ftell(filePtr);`
	`  fseek(filePtr, 0, SEEK_SET);`
	`  hexFileData = (LPBYTE) malloc(fileSize);`
	`  dataBuffer = (LPBYTE) malloc(LARGE_ENOUGH_TO_HOLD_THE_DATA);`
	`  if (hexFileData && dataBuffer)`
	`  {`
	`  bytesRead = fread((void *) hexFileData, 1, fileSize, filePtr);`
	`  if (bytesRead == fileSize)`
	`  {`
	`  dataSize = QD_UnFormatHexFileData(`
	`  (LPBYTE) hexFileData,`
	`  (LPBYTE) dataBuffer);`
	`  if (dataSize)`
	`  {`
	`  printf("The buffer contains %d raw data bytes\n",`
	`  dataSize);`
	`  }`
	`  }`
	`  free((void *) dataBuffer);`
	`  free((void *) hexFileData);`
	`  }`
	`  fclose(filePtr);`
	`}`
	
	Output:
	
	`The buffer contains 256 raw data bytes`

3.55. QD_WaitForReply 0x3E

Description	Checks the Receive Queue to ensure that it is ready for processing and that the specified number of bytes is residing in the queue for retrieval

Prototype	`DWORD QD_WaitForReply(`
	`  HANDLE unitHandle,`
	`  DWORD bytesExpected);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *bytesExpected*— In: Number of bytes that is expected to reside in the Receive Queue
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0021` *QD_ERROR_DEVICE_TIMED_OUT* `0x0022` *QD_ERROR_RECEIVE_QUEUE_OVERRUN*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE command[QD_COMMAND_BUFFER_SIZE];`
	`DWORD bytesTransmitted;`
	`DWORD commandLength;`
	`DWORD status;`
	
	`command[0] = 0x08;`
	`commandLength = 1;`
	`status = QD_Write(`
	`  unitHandle,`
	`  (LPBYTE) command,`
	`  commandLength,`
	`  (LPDWORD) &bytesTransmitted);`
	`if ((status == QD_SUCCESS) && (bytesTransmitted == commandLength))`
	`{`
	`  commandLength = QD_FIRMWARE_ID_LENGTH;`
	`  status = QD_WaitForReply(unitHandle, commandLength);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  status = QD_Read(`
	`  unitHandle,`
	`  (LPBYTE) command,`
	`  commandLength,`
	`  (LPDWORD) &bytesTransmitted);`
	`  if ((status == QD_SUCCESS) && (bytesTransmitted == commandLength))`
	`  {`
	`  printf("The firmware ID is %02X %02X %02X %02X\n",`
	`  command[0], command[1], command[2], command[3]);`
	`  }`
	`  }`
	`}`
	
	Output:
	
	`The firmware ID is 0D 16 01 02`

SPECIFICATION: QCOM INTERFACE DLL

3.56. QD_Write 0x3F

Description	Writes data to the specified module

Prototype	`DWORD QD_Write(`
	`  HANDLE unitHandle,`
	`  LPBYTE writeBuffer,`
	`  DWORD writeBufferSize,`
	`  LPDWORD bytesWritten);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *writeBuffer*— In: Buffer that contains the data to be written • *writeBufferSize*— In: Size in bytes of the write buffer • *bytesWritten*— Out: Pointer to the number of bytes actually written
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING*

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE command[QD_COMMAND_BUFFER_SIZE];`
	`DWORD bytesTransmitted;`
	`DWORD commandLength;`
	`DWORD status;`
	
	`command[0] = 0x08;`
	`commandLength = 1;`
	`status = QD_Write(`
	`  unitHandle,`
	`  (LPBYTE) command,`
	`  commandLength,`
	`  (LPDWORD) &bytesTransmitted);`
	`if ((status == QD_SUCCESS) && (bytesTransmitted == commandLength))`
	`{`
	`  commandLength = QD_FIRMWARE_ID_LENGTH;`
	`  status = QD_WaitForReply(unitHandle, commandLength);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  status = QD_Read(`
	`  unitHandle,`
	`  (LPBYTE) command,`
	`  commandLength,`
	`  (LPDWORD) &bytesTransmitted);`
	`  if ((status == QD_SUCCESS) && (bytesTransmitted == commandLength))`
	`  printf("The firmware ID is %02X %02X %02X %02X\n",`
	`  command[0], command[1], command[2], command[3]);`
	`  }`
	`}`
	
	Output:
	`The firmware ID is 0D 16 01 02`

SPECIFICATION: QCOM INTERFACE DLL

3.57. QD_WriteCoefficientDataToDevice 0x40

Description	Stores the coefficient data in transducer memory if a digital transducer with memory is attached, or in QCOM memory otherwise

Prototype	`DWORD QD_WriteCoefficientDataToDevice(`
	`  HANDLE unitHandle,`
	`  LPBYTE coefficientData);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *coefficientData*— In: 256-byte buffer of coefficient file data
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING* ‡ See the exception note for ***Next Byte*** in section 4

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	
	`status = QD_ReadCoefficientDataFromHexFile(`
	`  (LPBYTE) "XD242109.hex",`
	`  (LPBYTE) coefficientData);`
	`if (status == QD_SUCCESS)`
	`{`
	`  status = QD_WriteCoefficientDataToDevice(`
	`  unitHandle,`
	`  (LPBYTE) coefficientData);`
	`}`

SPECIFICATION: QCOM INTERFACE DLL

3.58. QD_WriteCoefficientDataToHexFile 0x41

Description	Saves the specified coefficient data to the specified file in hex file format

Prototype	`DWORD QD_WriteCoefficientDataToHexFile(`
	`  LPBYTE coefficientData,`
	`  LPBYTE pathName);`

Parameters	• *coefficientData*— In: 256-byte buffer of coefficient file data • *pathName*— In: Null-terminated string that includes the file name and the path name of the coefficient hex data file
Return Value	0x0000 *QD_SUCCESS* 0x0014 *QD_ERROR_MEMORY_ALLOCATION_FAILED* 0x0015 *QD_ERROR_FILE_OPEN_FAILURE* 0x0017 *QD_ERROR_FILE_SIZE_MISMATCH* 0x002A *QD_ERROR_NULL_POINTER_PARAMETER* 0x002B *QD_ERROR_ZERO_LENGTH_STRING_PARAMETER* 0x0070 *QD_ERROR_FILE_NOT_FOUND*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`char *filename = "CF_New.hex";`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	
	`status = QD_ReadCoefficientDataFromDevice(`
	`  unitHandle,`
	`  (LPBYTE) coefficientData);`
	`if (status == QD_SUCCESS)`
	`{`
	`  status = QD_WriteCoefficientDataToHexFile(`
	`  (LPBYTE) coefficientData,`
	`  (LPBYTE) filename);`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  printf("The coefficient hex file %s was successfully created\n",`
	`  fileName);`
	`  }`
	`}`
	
	Output:
	
	`The coefficient hex file CF_New.hex was successfully created`

3.59. QD_WriteCoefficientDataToModulePage 0x43

Description	Writes the specified coefficient data to the memory of the specified module

Prototype	`DWORD QD_WriteCoefficientDataToModulePage(`
	`  HANDLE unitHandle,`
	`  BYTE pageNumber,`
	`  LPBYTE coefficientData);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *pageNumber*— In: Memory page at which to write the data, as follows: `0x00` Page at offset 0x0000 `0x01` Page at offset 0x0100 `0x02` Page at offset 0x0200 `0x03` Page at offset 0x0300 • *coefficientData*— In: 256-byte buffer of coefficient file data
Return Value	`0x0000` *QD_SUCCESS* `0x0001` *QD_ERROR_INVALID_HANDLE* `0x0003` *QD_ERROR_RECEIVE_QUEUE_NOT_READY* `0x0004` *QD_ERROR_WRITE_ERROR* `0x0006` *QD_ERROR_INVALID_PARAMETER* `0x0007` *QD_ERROR_INVALID_REQUEST_LENGTH* `0x0008` *QD_ERROR_DEVICE_IO_FAILED* `0x000D` *QD_ERROR_READ_TIMED_OUT* `0x000E` *QD_ERROR_WRITE_TIMED_OUT* `0x000F` *QD_ERROR_IO_PENDING* ‡ See the exception note for ***Next Byte*** in section 4

Example	**Code:**
	`#include "qdUSB.h"`
	`. . .`
	`BYTE coefficientData[QD_COEFFICIENT_DATA_SIZE];`
	`DWORD status;`
	
	`status = QD_ReadCoefficientDataFromHexFile(`
	`  (LPBYTE) "XD242109.hex",`
	`  (LPBYTE) coefficientData);`
	`if (status == QD_SUCCESS)`
	`{`
	`  status = QD_WriteCoefficientDataToModulePage(`
	`  unitHandle,`
	`  2,`
	`  (LPBYTE) coefficientData);`
	`}`

3.60. QD_WriteFirmwarePage 0x45

Description	Writes the specified firmware data to specified firmware page of the module **Note: This function must only be called in Boot Loader Mode**

Prototype	`DWORD QD_WriteFirmwarePage(`
	`  HANDLE unitHandle,`
	`  LPBYTE firmwareData,`
	`  BYTE pageNumber,`
	`  LPWORD pageCRC);`

Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open*** • *firmwareData*— In: 64512 (63 X 1024)-byte buffer of valid firmware data • *pageNumber*— In: Firmware page number at which to write the firmware data • *pageCRC*— Out: Firmware page CRC returned following writing of the firmware page
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE lastPage = QD_FIRMWARE_LAST_PAGE_C8051F320;`
	`BYTE pageNumber;`
	`BYTE firmwareData[QD_FIRMWARE_MAXIMUM_DATA_SIZE];`
	`WORD pageCRC;`
	`DWORD status;`
	
	`QD_ReadFirmwareDataFromFile(`
	`  (LPBYTE) "FW0752.hex",`
	`  (LPBYTE) firmwareData);`
	`QD_SetFirmwareKeyCodes(unitHandle, QD_FIRMWARE_KEY_0, QD_FIRMWARE_KEY_1);`
	`status = EraseFirmwarePage(`
	`  unitHandle,`
	`  lastPage);`
	`if (status == QD_SUCCESS)`
	`{`
	`  for (pageNumber = QD_FIRMWARE_FIRST_PAGE_C8051F320;`
	`  ((pageNumber <= lastPage) && (status == QD_SUCCESS)); pageNumber++)`
	`  {`
	`  pageCRC = 0;`
	`  status = QD_WriteFirmwarePage(`
	`  unitHandle,`
	`  (LPBYTE) firmwareData,`
	`  pageNumber,`
	`  (LPWORD) &pageCRC);`
	`  }`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  QD_WriteFirmwareSignature(unitHandle);`
	`  QD_SetFirmwareKeyCodes(unitHandle, 0, 0);`
	`  }`
	`}`

SPECIFICATION: QCOM INTERFACE DLL

3.61. QD_WriteFirmwareSignature 0x46

Description	Writes the firmware signature to the specified module **Note: This function must only be called in Boot Loader Mode**
Prototype	`DWORD QD_WriteFirmwareSignature(` `  HANDLE unitHandle);`
Parameters	• *unitHandle*— In: Module handle returned by ***QD_Open***
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING* 0x0023 *QD_ERROR_FIRMWARE_SIGNATURE_NOT_ERASED*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`BYTE lastPage = QD_FIRMWARE_LAST_PAGE_C8051F320;`
	`BYTE pageNumber;`
	`BYTE firmwareData[QD_FIRMWARE_MAXIMUM_DATA_SIZE];`
	`WORD pageCRC;`
	`DWORD status;`
	
	`QD_ReadFirmwareDataFromFile(`
	`  (LPBYTE) "FW0752.hex",`
	`  (LPBYTE) firmwareData);`
	`QD_SetFirmwareKeyCodes(unitHandle, QD_FIRMWARE_KEY_0, QD_FIRMWARE_KEY_1);`
	`status = EraseFirmwarePage(`
	`  unitHandle,`
	`  lastPage);`
	`if (status == QD_SUCCESS)`
	`{`
	`  for (pageNumber = QD_FIRMWARE_FIRST_PAGE_C8051F320;`
	`  ((pageNumber <= lastPage) && (status == QD_SUCCESS)); pageNumber++)`
	`  {`
	`  pageCRC = 0;`
	`  status = QD_WriteFirmwarePage(`
	`  unitHandle,`
	`  (LPBYTE) firmwareData,`
	`  pageNumber,`
	`  (LPWORD) &pageCRC);`
	`  }`
	`  if (status == QD_SUCCESS)`
	`  {`
	`  QD_WriteFirmwareSignature(unitHandle);`
	`  QD_SetFirmwareKeyCodes(unitHandle, 0, 0);`
	`  }`
	`}`

3.62. QD_GetMemoryType 0x1D

Description	Retrieves the memory type, if present, of the target device

Prototype	`DWORD QD_GetMemoryType(`
	`  HANDLE unitHandle,`
	`  BYTE device,`
	`  LPBYTE memoryType);`

Parameters	• *unitHandle* — In: Module handle returned by ***QD_Open*** • *device* — In: Target device of the memory type request, as follows: *QD_DEVICE_TRANSDUCER* 0x00 *QD_DEVICE_MODULE* 0x01 • *memoryType* — Out: Pointer to the memory type, as follows: 0x00 Memory is absent in the target device 0x01 Memory is FRAM 0x02 Memory is EEPROM
Return Value	0x0000 *QD_SUCCESS* 0x0001 *QD_ERROR_INVALID_HANDLE* 0x0003 *QD_ERROR_RECEIVE_QUEUE_NOT_READY* 0x0004 *QD_ERROR_WRITE_ERROR* 0x0006 *QD_ERROR_INVALID_PARAMETER* 0x0007 *QD_ERROR_INVALID_REQUEST_LENGTH* 0x0008 *QD_ERROR_DEVICE_IO_FAILED* 0x000D *QD_ERROR_READ_TIMED_OUT* 0x000E *QD_ERROR_WRITE_TIMED_OUT* 0x000F *QD_ERROR_IO_PENDING*

Example	**Code:**
	
	`#include "qdUSB.h"`
	`. . .`
	`DWORD status;`
	`BYTE memoryType;`
	
	`status = QD_GetMemoryType(`
	`  unitHandle,`
	`  QD_DEVICE_TRANSDUCER,`
	`  (LPBYTE) &memoryType);`
	`if (status == QD_SUCCESS)`
	`{`
	`  printf("The transducer memory type is %d\n", memoryType);`
	`}`
	
	Output:
	
	`The transducer memory type is 4`

4. ERROR CODES:

Each status value returned from DLL functions contains three sections of information: the Upper Byte, the Next Byte, and the Lower Word. For example, if status `0x34050016` was detected, it indicates that the function ***QD_ReadUnitADC*** encountered an error in which the read of the ADC was successful, but the contents of the ADC register was zero.

All status values returned from DLL functions are 32-bit error codes that are formatted as follows:

Upper Byte (bits 24 through 31)

Function number if the ***Lower Word*** is nonzero; otherwise, it is zero. The function numbers are listed in this document on the title lines of the corresponding DLL functions that return them.

Next Byte (bits 16 through 23)

Relative problem locus (location) within the function, and can differ from one function to the next

The exception to this occurs in the functions

QD_ReadCoefficientDataFromDevice
QD_ReadCoefficientDataFromModulePage
QD_ReadCoefficientDataFromSourcePage
QD_WriteCoefficientDataToDevice
QD_WriteCoefficientDataToModulePage
QD_WriteCoefficientDataToTargetPage

For these functions, the ***Next Byte*** reflects a bitmap of the target device and 256-byte memory page in the device if the ***Lower Word*** is nonzero; otherwise, it is zero. The bitmap convention is as follows:

The highest two bits: the device (00b = the transducer, 01b = the QCOM module)
The lower six bits: the memory page number (0x00 through 0x3F, inclusive)

Lower Word (bits 0 through 15)

The actual error code, as listed under ***Return Value*** in each function description, defined as follows:

`0x0000`	*QD_ERROR_NO_ERROR* or *QD_SUCCESS*
`0x0001`	*QD_ERROR_INVALID_HANDLE*
`0x0002`	*QD_ERROR_READ_ERROR*
`0x0003`	*QD_ERROR_RECEIVE_QUEUE_NOT_READY*
`0x0004`	*QD_ERROR_WRITE_ERROR*
`0x0005`	*QD_ERROR_RESET_ERROR*
`0x0006`	*QD_ERROR_INVALID_PARAMETER*
`0x0007`	*QD_ERROR_INVALID_REQUEST_LENGTH*
`0x0008`	*QD_ERROR_DEVICE_IO_FAILED*
`0x0009`	*QD_ERROR_INVALID_DATA_RATE*

SPECIFICATION: QCOM INTERFACE DLL

0x000A	*QD_ERROR_FUNCTION_NOT_SUPPORTED*
0x000B	*QD_ERROR_GLOBAL_DATA_ERROR*
0x000C	*QD_ERROR_SYSTEM_ERROR*
0x000D	*QD_ERROR_READ_TIMED_OUT*
0x000E	*QD_ERROR_WRITE_TIMED_OUT*
0x000F	*QD_ERROR_IO_PENDING*
0x0010	*QD_ERROR_CHECKSUM_ERROR*
0x0011	*QD_ERROR_XD_PRESSURE_COUNT_ZERO*
0x0012	*QD_ERROR_XD_TEMPERATURE_COUNT_ZERO*
0x0013	*QD_ERROR_XD_BOTH_COUNTS_ZERO*
0x0014	*QD_ERROR_MEMORY_ALLOCATION_FAILED*
0x0015	*QD_ERROR_FILE_OPEN_FAILURE*
0x0016	*QD_ERROR_ADC_COUNT_ZERO*
0x0017	*QD_ERROR_FILE_SIZE_MISMATCH*
0x0018	*QD_ERROR_INVALID_HEX_FILE_DATA*
0x0019	*QD_ERROR_INCORRECT_DATA_SIZE*
0x001A	*QD_ERROR_INVALID_FIRMWARE_PAGE*
0x001B	*QD_ERROR_UNIT_NOT_READY*
0x001C	*QD_ERROR_CADENCE_TIMER_ZERO*
0x001D	*QD_ERROR_DATA_RATE_ZERO*
0x001E	*QD_ERROR_INVALID_I2C_REPLY*
0x001F	*QD_ERROR_FIRMWARE_ID_ZERO*
0x0020	*QD_ERROR_DATA_TRANSFER_FAILURE*
0x0021	*QD_ERROR_DEVICE_TIMED_OUT*
0x0022	*QD_ERROR_RECEIVE_QUEUE_OVERRUN*
0x0023	*QD_ERROR_FIRMWARE_SIGNATURE_NOT_ERASED*
0x0024	*QD_ERROR_XD_VOLTAGE_TOO_HIGH*
0x0025	*QD_ERROR_XD_VOLTAGE_TOOL_LOW*
0x0026	*QD_ERROR_XD_CURRENT_TOO_HIGH*
0x0027	*QD_ERROR_XD_CURRENT_TOO_LOW*
0x0028	*QD_ERROR_WRITE_ACKNOWLEDGE_FAILED*
0x0029	*QD_ERROR_UNREACHABLE_CONDITION*
0x002A	*QD_ERROR_NULL_POINTER_PARAMETER*
0x002B	*QD_ERROR_ZERO_LENGTH_STRING_PARAMETER*
0x0040	*QD_ERROR_INVALID_COEFFICIENT_DATA*
0x0070	*QD_ERROR_FILE_NOT_FOUND*
0x0071	*QD_ERROR_INVALID_FILE_NAME*
0x0072	*QD_ERROR_INVALID_FILE_FOUND*
0x00FF	*QD_ERROR_NO_DEVICE_FOUND*