**Overview:** This document documents the low-level interface specification for the NEOS CAN adapter. The NEOS CAN adapter is a specialized interface to allow one to monitor/generate messages on a NEOS system. The uses a specialized bit rate that is not supported by most CAN bus interfaces to allow audio to be streamed over the bus. The bitrate for the NEOS bus is 1024000Bits/Second and uses 11-bit CAN IDs and 8-bit messages.

**I/O:** The core NEOS CAN adapter circuitry is identical to the microprocessor circuitry on a NEOS node. This circuitry is housed in a Deutsch EEC-325X4B-E016 enclosure and provides 3 I/Os. There are two 4-pin Deutsch connectors (Male/Female pair) for insertion into the CAN bus loop and a short male USB type-A connector. The USB connector was design for connecting to a USB Male-A to Female-A extension cable. Also note that the adapter can be internally or externally terminated if needed. There is a 100ohm termination resistor that can be enabled via software. By default (after reset), this termination is not enabled.

**USB Interface:**  The USB interface is provided by an FTDI FT245R USB to FIFO interface. FTDI ([www.ftdichip.com](http://www.ftdichip.com)) provides an interface DLL that allows one to easily read and write to a set of FIFOs that are implemented on the FT245 IC. There are code examples for most languages on the FTDI website. From the PC side, application software will have simple read/write commands to access the input/output FIFOs on the FT245 IC. Figure 1 illustrates this flow.

**Figure 1: NEOS CAN Adapter Interface Model**

To send CAN data, the PC application has to wrap the CAN messages with a simple wrapper and place them in the USB output FIFO. When CAN messages are received, the microcontroller on the adapter will wrap the messages and place them in the USB I/O queue. The NEOS CAN interface also implements some simple “control” messages that can be used to configure the adapter (CAN Message Filtering, etc).

**Message Transport:** All messages going to and coming from the NEOS CAN adapter must be transported with a simple wrapper. The messages (both CAN and control) are simple binary structures that are the “payload” in the message transport. All fields in the structure are little-endian byte ordered and are byte packed. The pseudo-structure below shows the message transport.

//Data Type Declarations

#define BYTE unsigned char //BYTES are 8-bits

#define WORD unsigned short //WORDs are 2 bytes

#define DWORD unsigned long //DWORDs are 4 bytes

#define BOOL BYTE //Booleans are bytes where 0 is false and non-zero if true

struct

{

WORD Header = 0xAAAA; // Message Start/Sync Header

WORD PayloadLength; //This the Length of the Payload only

BYTE MessagePayload[PayloadLength]; //Byte array of the Payload

WORD CRC; //16-bit CRC CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1)

//The CRC is calculated on entire message frame

// (Header+Length+Payload)

} MessageTransport;

//Example Routine for calculating the

//CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1)

WORD CalcCRC(BYTE \*data, WORD length)

{

crc = 0xffff;

for (i=0;i<length;i++)

{

crc = (unsigned char)(crc >> 8) | (crc << 8);

crc ^= data[i];

crc ^= (unsigned char)(crc & 0xff) >> 4;

crc ^= (crc << 8) << 4;

crc ^= ((crc & 0xff) << 4) << 1;

}

return crc;

}

**Messages Implemented:** This section enumerates the messages implemented on the NEOS CAN Adapter.A couple notes:

* The messages are the payload to be wrapped as defined in the messaging transport section
* All interface messages have a message identifier that is the first byte of the message structure.
* The message structures are documented in a ‘pseudo C’ format.

**Message:** **CAN\_INTERFACE\_PING**

**Message Code:** 0x00

**Description:**  This message is used for testing purposes. Sending this message to the CAN interface adapter will trigger the firmware to respond with a CAN\_INTERFACE\_PONG message.

**Message Structure:**

#define CAN\_INTERFACE\_PING 0x00

struct

{

BYTE MessageId = CAN\_INTERFACE\_PING;

} CANInterfacePing;

**Message:** **CAN\_INTERFACE\_PONG**

**Message Code:** 0x00

**Description:**  This message is used for testing purposes. This message is sent in response to a CAN\_INTERFACE\_PING

**Message Structure:**

#define CAN\_INTERFACE\_PONG 0x11

struct

{

BYTE MessageId = CAN\_INTERFACE\_PONG;

} CAN

**Message:** **CAN\_INTERFACE\_RESET**

**Message Code:** 0x00

**Description:**  This message resets the firmware in the CAN interface adapter. This also disables the internal CAN termination and sets the CAN message exclusion range from 0x100 to 0x1ff (Audio Message Space)

**Message Structure:**

#define CAN\_INTERFACE\_RESET 0x01

struct

{

BYTE MessageId = CAN\_INTERFACE\_RESET;

} CANInterfaceReset;

**Message:** **SET\_ID\_EXCLUDE\_RANGE**

**Message Code:** 0x05

**Description:**  This sets the internal exclusion filter for the CAN interface. Messages within this range will be ignored and not passed to the USB interface. After reset all CAN messages with the ID from 0x100 to 0x1FF are excluded. Most of the messages on the NEOS bus are audio messages from the master controller. For most applications monitoring these messages is not needed and simplifies the PC software as the host program will not have to deal with the rapid audio messages.

**Message Structure:**

#define SET\_ID\_EXCLUDE\_RANGE 0x05

struct

{

BYTE MessageId = SET\_ID\_EXCLUDE\_RANGE;

DWORD IDExcludeLow; //Lower bound of the CAN message exclude range

DWORD IDExcludeHigh; //Upper Bound of the CAN message exclude range

} SetIDExcludeRange;

**Message:** **SET\_CAN\_TERMINATION**

**Message Code:** 0x06

**Description:**  This message can enable/disable termination on the CAN transceiver inside the adapter. A termination resistor of 100ohms can be switched in and out if needed.

**Message Structure:**

#define SET\_CAN\_TERMINATION 0x06

struct

{

BYTE MessageId = SET\_CAN\_TERMINATION;

BOOL EnableInternalTermination; // 0x00 (false No Internal Termination

// non-zero (true) --> Can transciever

// on adapter terminated to 100ohms

} SetCANTermination;

**Message:** **TX\_CAN\_MSG**

**Message Code:** 0x03

**Description:**  This message is used to put a CAN message on the bus from the USB interface. All messages must be 8-bytes in length. CAN ID is 11-bit and is right justified in the DWORD (lower 11-bits for the ID).

**Message Structure:**

#define TX\_CAN\_MSG 0x03

struct

{

BYTE MessageId = TX\_CAN\_MSG;

DWORD CANId; // 11-bit CAN ID (Right Justified in DWORD)

BYTE CanData[8]; // 8-bytes of CAN Data

} TXCanMsg;

**Message:** **RX\_CAN\_MSG**

**Message Code:** 0x04

**Description:**  This message is sent to the USB interface when a new CAN message is received from the bus. All messages are 8-bytes in length. CAN ID is 11-bit and is right justified in the WORD (lower 11-bits for the ID).

**Message Structure:**

#define RX\_CAN\_MSG 0x04

struct

{

BYTE MessageId = TX\_CAN\_MSG;

DWORD CANId; // 11-bit CAN ID (Right Justified in DWORD)

BYTE CanData[8]; // 8-bytes of CAN Data

} TXCanMsg;