Dexbridge2

Wixel code that can act as a bridge between a Dexcom G4 Transmitter and a smart phone using Bluetooth 4.0 (BLE). Requires that the Wixel be connected to a HM-10 module, using the design originally put together by Stephen Black for his DexDrip system. There is a minor hardware modification required in order to receive the bridge battery voltage, for monitoring in the app.

The original wixel code for DexDrip was based on work by Adrien de Croy, Lorelai Lane, and others, but it has some limitations. This bridge code has the following features:

- Does not receive any Dexcom packets until it has been given a Dexcom G4 Transmitter ID to filter on. This is important to ensure that it correctly locks on to the signal of the transmitter in question, and only passes that Transmitter ID's data to the phone app.
- Stores the Transmitter ID in flash, so that it survives if power fails for any reason. The app does not need to reset this in such an event.
- Sends a "beacon" to the app when it wakes up from low power mode, indicating the Transmitter ID it is filtering on.
- Accepts a Transmitter ID packet from the phone app, and saves it to flash. Note, the phone app must await either a data packet or a beacon packet before determining if the Transmitter ID is incorrect, and sending an ID packet to the bridge.
- Sends the Bridge battery voltage as part of the data packet.
- Automatically corrects the packet "listen window" for changes in overall program cycle time. This ensures that any changes to the code does not require any further experimentation to the listen window to make it work reliably.
- Sets the HM-10 module's BLE ID to "DB-XXXXXXXXXXXX", where XXXXXXXXXXX is the wixel serial number. This ensures that each bridge has a unique ID, making the BLE connection more reliable.

Protocol

Each packet of data sent or received by the bridge is described below. Common to each packet are the first two 8bit bytes. The first byte is the length of the packet in bytes. The second is an ID for the type of packet being sent.

Data Packet

A Data packet is sent by the wixel to the phone app. It contains the relevant data sent from the Dexcom G4 Transmitter, plus the bridge battery voltage and TxID it is filtering on.

The data packet has the following structure:

Value	Data Type	Description
Packet Length (17)	8 bit unsigned integer	Number of bytes in the packet
0x00	8 bit unsigned integer	Code for Data Packet
Raw Signal	32 bit unsigned integer	Raw Sensor signal
Filtered Signal		Filtered Sensor signal
Dexcom Tx Battery Voltage	8 bit unsigned integer	The Transmitter battery voltage. Usually around 214

		for a new transmitter. The app should alert if this reaches <= 207, that the transmitter requires replacement.
Bridge Battery Voltage	16 bit unsigned integer	The bridge batter voltage.
Dexcom TxID	32 bit unsigned integer	Encoded Dexcom Transmitter ID that the bridge is filtering on.

Upon receiving this packet, the phone app has to process it, taking the parts of the packet it will use.

If the app determines that the Dexcom TxID is different to it's own setting, it should immediately send a TXID packet back to the bridge, and ignore the packet.

If the app is happy with the Dexcom TxID sent, it should accept the packet and immediately send back an acknowledgement packet. The acknowledgement packet will immediately tell the wixel to go into low power mode.

The acknowledgement packet structure is as follows:

Value	Data Type	Description
Packet Length (0x02)	8 bit unsigned integer	Number of bytes in the packet
0xF0	8 bit unsigned integer	Code for Data Packet

Note that the wixel will otherwise go into low power mode if it does not receive an acknowledgement or TXID packet within 3 seconds of transmitting a data packet.

TXID packet

The TXID packet is sent from the phone app to the bridge to set the bridge to filter on a single Dexcom G4 transmitter ID. This is important to ensure the bridge correctly "locks" to the correct transmitter for a patient, and also to ensure the app only receives packets from the transmitter of the patient it is monitoring.

The structure of the TXID packet is as follows:

Value	Data Type	Description
Packet Length (0x06)	8 bit unsigned integer	Number of bytes in the packet
0x01	8 bit unsigned integer	Code for Data Packet
TxID	32 bit unsigned integer	Encoded 32 bit integer representing the Dexcom G4 Transmitter ID that the bridge is filtering packets on.

Beacon packet

The Beacon packet is sent from the bridge to the phone app to indicate which Dexcom G4 Transmitter ID it is filtering on. The app can use this beacon to know when the bridge is active, and if it has a different Transmitter ID to what the app is configured for, it can correct this by sending a TXID packet.

The structure of the Beacon packet is as follows:

Value	Data Type	Description
Packet Length (0x06)	8 bit unsigned integer	Number of bytes in the packet
0xF1	8 bit unsigned integer	Code for Data Packet
TxID	32 bit unsigned integer	Encoded 32 bit integer representing the Dexcom G4 Transmitter ID that the bridge should filter packets on.

Note, this packet also doubles as the acknowledgement packet for a TXID packet. When the app receives this packet it can be sure that this is the Transmitter ID value set in the wixel flash memory.

Decoding and Encoding a Transmitter ID Long Int

In order for the app to send the correct value in a TXID packet to the bridge, you need to be able to encode the text of the Transmitter ID to a long int. This is done using the following pseudo code, taken directly from the original dexbridge code. Your app will need to replicate this process in order to send the correct data.

```
char SrcNameTable[32] = { '0', '1', '2', '3', '4', '5', '6', '7',
                          '8', '9', 'A', 'B', 'C', 'D', 'E', 'F',
                          'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P',
                          'Q', 'R', 'S', 'T', 'U', 'W', 'X', 'Y' };


/* asciiToDexcomSrc - function to convert a 5 character string into
a unit32 that equals a Dexcom transmitter Source address.  The 5
character string is equivalent to the characters printed on the
transmitter, and entered into a receiver.
      Parameters:
            addr -     a 5 character string. eg "63GEA"
      Returns:
            uint32-    a value equivalent to the incodeded Dexcom
Transmitter address.
Uses:
      getSrcValue(char)
      This function returns a value equivalent to the character for
encoding.
      See srcNameTable[]
*/
uint32 asciiToDexcomSrc(char addr[6])
{
      // prepare a uint32 variable for our return value
      uint32 src = 0;
      // look up the first character, and shift it 20 bits left.
      src |= (getSrcValue(addr[0]) << 20);
      // look up the second character, and shift it 20 bits left.
      src |= (getSrcValue(addr[1]) << 15);
      // look up the third character, and shift it 20 bits left.
      src |= (getSrcValue(addr[2]) << 10);
```

```
	// look up the fourth character, and shift it 20 bits left.
	src |= (getSrcValue(addr[3]) << 5);
	// look up the fifth character, and shift it 20 bits left.
	src |= getSrcValue(addr[4]);
	//printf("asciiToDexcomSrc: val=%u, src=%u\r\n", val, src);
	return src;
}

/* getSrcValue  -	function to determine the encoding value of a
character in a Dexcom Transmitter ID.
Parameters:
	srcVal	-	The character to determine the value of
Returns:
	uint32	-	The encoding value of the character.
*/
uint32 getSrcValue(char srcVal)
{
	uint8 i = 0;
	for(i = 0; i < 32; i++)
	{
		if (SrcNameTable[i]==srcVal) break;
	}
	//printf("getSrcVal: %c %u\r\n",srcVal, i);
	return i & 0xFF;
}
```

Decoding a long integer transmitter ID is far simpler. You may implement a similar piece of code if you are storing the ID as a long int, but wish to display the text equivalent.

```
// convert the passed uint32 Dexcom source address into an ascii
string in the passed char addr[6] array.
void dexcom_src_to_ascii(uint32 src, char addr[6])
{
	//each src value is 5 bits long, and is converted in this way.
	addr[0] = SrcNameTable[(src >> 20) & 0x1F];	//the last
character is the src, shifted right 20 places, ANDED with 0x1F
	addr[1] = SrcNameTable[(src >> 15) & 0x1F];	//etc
	addr[2] = SrcNameTable[(src >> 10) & 0x1F];	//etc
	addr[3] = SrcNameTable[(src >> 5) & 0x1F];	//etc
	addr[4] = SrcNameTable[(src >> 0) & 0x1F];	//etc
	addr[5] = 0;	//end the string with a null character.
}
```