DOCUMENT NAME: BleConnector Cli Documentation	
Version: 1.3	
FILE NAME: BleConnector_Documentation	Update: 2022/09/11

Revision History:

#	Date	Reviser	Update	Version
1	2022/08/30	M. M. Hejazi	Primary Edition	1.0
2	2022/09/08	M. M. Hejazi	Blood Pressure & Weight Scale support	1.1
3	2022/09/10	M. M. Hejazi	Stethoscope support	1.2
4	2022/09/11	M. M. Hejazi	Added settings.json	1.3

Contact us:

Name: Mohammad Mahdi Hejazi

Phone: +98 914 650 1380

Email: Hejazi.m.mahdi@gmail.com

Table of Contents

1. General 4
1.1. Overview 4
1.2. Supported Devices 4
1.3. Usage 4
1.4. Commands 4
2. Program 5
2.1. Program.cs file 5
3. Ble Package 7
3.1. Core 7
3.2. Interface 7
3.3. Manager 8
3.3.1. Glucometer 8
3.3.2. Oximeter 8
3.3.3. Thermometer 8
3.3.4. Blood Pressure 9
3.3.5. Weight Scale 9
3.3.6. Stethoscope 9
4. Models 10
4.1. DeviceTypes 10
4.2. ErrorCodes 10
4.3. Range 11
4.4. Timestamp 11
4.5. GlucometerMeasurement 12
4.6. OximeterMeasurement 13
4.7. ThermometerMeasurement 13
4.8. BloodPressureMeasurement 13
4.9. WeightMeasurement 14
4.10. StethoscopeMeasurement 15
4.11. EchoMode 15
5. Utils 16

5.1. DataUtil 16
5.2. DecodeAudioData 16
5.3. DecodeAudioDataUtils 16
5.4. HandleRawData 16

1. General

1.1. Overview

This documentation describes information about BleConnector command-line tool which will be used to connect and communicate with pre-defined medical devices via “Bluetooth Low Energy”.

This cli will be used in the user-side application to establish a connection with the desired peripheral and retrieve its measurements.

**Note*: For this cli to work, this host computer should have internal or external Bluetooth 4.0+ module.*

1.2. Supported Devices

Here is a list of supported devices.

Device Type	Brand	Model
Glucometer	Beurer	GL50 evo
Oximeter	Beurer	PO60
Thermometer	Beurer	FT95
Blood Pressure	Beurer	BM67
Weight Scale	Beurer	BF70
Stethoscope	Mintti	Smartho-D2

**This list will update when a new device gets supported. **

1.3. Usage

To use the cli, first we need to run it inside a new command-line. When running, we must pass the required arguments and wait to get its result. The result is either the preferred data in json format (more about formats in “Models”) or an error code.

1.4. Commands

Tis cli follows the standard run palette for its commands as below:

```
BleConnector.exe <Command> <args>
```

**Currently no flags are defined for this cli. **

Here is a list of valid commands:

Command	Arguments	Description

Glucometer	Target glucometer device mac address	Tries to connect to glucometer peripheral with the given mac address and get its latest measurement.
Oximeter	Target oximeter device mac address	Tries to connect to oximeter peripheral with the given mac address and get its latest measurement.
Thermometer	Target thermometer device mac address	Tries to connect to thermometer peripheral with the given mac address and get its latest measurement.
BloodPressure	Target blood pressure device mac address	Tries to connect to thermometer peripheral with the given mac address and get its latest measurement.
WeightScale	Target weight scale device mac address	Tries to connect to thermometer peripheral with the given mac address and get its latest measurement.
Stethoscope	Target weight scale device mac address	Tries to connect get the encoded raw audio data and dumps the decoded audio data into a valid audio file.

Sample command:

```
> BleConnector.exe Glucometer f7:4c:87:32:62:ff
```

**Mac address is a unique identifier assigned to your medical device that can be used to scan the environment for that specific hardware and communicate with it. **

2. Program

2.1. Program.cs file

This file is the entry point for our program that handles the commands entered by the user.

When a command is entered by the user, gets evaluated and if it is correct, the command gets executed. Different stages of evaluation:

1. Check command format (as above)
2. Check command validity
3. Validate the entered mac address
4. Scan the environment for the device
5. Check if connection is established
6. Check if device is paired

If the command passes all of these tests, then the cli connects to the target device and starts the data transmission.

3. Ble Package

In this section all the files inside “Ble” package are explained.

3.1. Core

This file contains the “BleWatcher” variable as a singleton which is used to scan the environment, connect and communicate with peripherals.

List of different functions:

- **CheckBleSupport:** This functions check to see if the host device contains Bluetooth module (internal or external).
- **CheckBleEnabled:** This function checks to see if the host device has enabled its bluetooth.
- **Scan:** This functions gets the target device’s mac address and scans the environment for that device with the defined amount of timeout time *(default = 10s)*. When the device is not found at the timeout finished, scan stopes.
- **StartWatcher:** Starts the watcher to scan the environment.
- **OnScanResult:** This is a callback function and gets called whenever a new device is found in the scan operation. Then it checks the device to see if it is our desired device or not.
- **OnScanError:** This is a callback functions and gets called whenever an error occurs in the scan operation.

3.2. Interface

This file contains all the functions used to communicate with the peripheral.

List of different functions:

- **SetDevice:** Gets a BluetoothLEDevice as input and sets it as its target to communicate with.
- **Subscribe:** This function is given a characteristic uuid and a callback function and tries to subscribe to the characteristics with the given callback.
- **Unsubscribe:** This function is given a characteristic uuid and a callback function and tries to unsubscribe the callback from characteristic.
- **WriteData:** Gets a characteristic uuid and data in format of bytes and tries to write the data on the characteristic if supported.
- **ReadData:** Gets a characteristic uuid and tries to read the data from that characteristic if supported.
- **CheckConnection:** Checks to see if the connection is ongoing and the device is still connected.

- **GetCharacteristic:** Iterates through all of the characteristic and returns the desired characteristic if available.
- **GetAllCharacteristics:** Iterates through all of the peripherals services and reads their different characteristic.

3.3. Manager

Manages the flow of data between the cli and the peripheral by executing the pre-defined instructions.

3.3.1. Glucometer

Gets the latest measurement in 7 steps:

- Subscribes to the "Access Control Characteristic" defined by the documentation.
- Subscribes to the "Measurement Characteristic" defined by the documentation.
- Writes the "0x01-06" command to the "Access Control Characteristic" to get the latest measurement.
- Callback function gets data from the peripheral and parses it to a valid measurement and prints it.
- Delays for 5 seconds.
- Unsubscribes from the "Access Control Characteristic".
- Unsubscribes from the "Measurement Characteristic".

3.3.2. Oximeter

Gets the latest measurement in 6 steps:

- Subscribes to the "Measurement Characteristic" defined by the documentation.
- Writes the "0x99-00-19" command to the defined characteristic to get the latest measurement.
- Callback function gets data from the peripheral in form of 12 packets of 20 bytes. Then the cli writes a response meaning "All data received. Send the next set of data" and append all the data together.
- Delays for 5 seconds.
- Unsubscribes from the "Measurement Characteristic".
- Splits all the data received from the peripheral to chunks of 24 bytes and parses each one of them to a valid measurement. Then prints the latest measurement according to the measurements' timestamps.

3.3.3. Thermometer

Gets the latest measurement in 5 steps:

- Subscribes to the "Measurement Characteristic" defined by the documentation.
- Callback function gets data from the peripheral and parses it to a valid measurement.
- Delays for 10 seconds.

- Unsubscribes from the “Measurement Characteristic”.
- Prints the latest record received from the peripheral.

3.3.4. Blood Pressure

Gets the latest measurement in 5 steps:

- Subscribes to the “Measurement Characteristic” defined by the documentation.
- Callback function gets data from the peripheral and parses it to a valid measurement.
- Delays for 5 seconds.
- Unsubscribes from the “Measurement Characteristic”.
- Prints the latest record received from the peripheral.

3.3.5. Weight Scale

Gets the latest measurement in 4 steps:

- Subscribes to the “Measurement Characteristic” defined by the documentation.
- Callback function gets data from the peripheral and parses it to a valid measurement. In the meantime, print the received value to get the live update of the scale.
- Waits until measurement is finished (defined by the packet size).
- Prints the latest record received from the peripheral.

**Not that if “UpdateWeight” field is set to true in “Settings”, while receiving the measurement it will print it to the console. **

3.3.6. Stethoscope

Gets the decoded audio data in 5 steps:

- Initialize necessary algorithms inside MinttiAlgo.dll file.
- Create / Open the output audio file (format can be .wav or .mp3) and write header inside it.
- Subscribe to “Stethoscope Mode Characteristic” to update the recording mode as it gets changed.
- Subscribe to “Measurement Characteristic” and receive the raw encoded audio data in form of notifications.
- Decode the data as it gets received and dump them inside the output file.
- Delay for the defined time (audioLengthSeconds) to record the required data.
- Unsubscribe from “Stethoscope Mode Characteristic”.
- Unsubscribe from “Stethoscope Measurement Characteristic”.
- Print the latest recorded audio file address according to “StethoscopeMeasurement” model.

4. Models

In this section all the models inside the "Model" package is explained. These models are used to parse the response from peripherals to valid measurements and print them inside the console.

4.1. DeviceTypes

This model is an enumerated data type, used to store different supported device types and also to distinguish different commands from each other.

Different values:

- **Glucometer**
- **Oximeter**
- **Thermometer**
- **BloodPressure**
- **WeightScale**
- **Stethoscope**

**This model has been used in "Program.cs" to validate the commands. **

4.2. ErrorCodes

This model is an enumerated data type, used to store different errors that may occur when the cli is running.

Different values and their meanings:

**First 10 codes are Bluetooth internal errors that happens when windows apis fail. **

- **ConsentRequired:** Consent from the user is required
- **DeviceNotConnected:** Target device is not connected.
- **DisabledByPolicy:** Bluetooth is disables/turned off by policy.
- **DisabledByUser:** Bluetooth is disables/turned off by user.
- **NotSupported:** Bluetooth is not supported by the OS.
- **OtherError:** Other errors.
- **RadioNotAvailable:** No Bluetooth hardware is found.
- **ResourceInUse:** Bluetooth is in use by another program.
- **Success:** No errors
- **TransportNotSupported:** Data transportation is not supported.

**These 5 codes are generated by the cli in case of failure in connecting stage. **

- **DeviceNotFound:** Device with the given mac address couldn't be found.
- **PairingRequired:** Pairing is needed to communicate with the device.
- **InvalidCommand:** Command is invalid.

- **InvalidMacAddress:** The given mac address doesn't follow the standard format.
- **InvalidDeviceType:** Given device type is invalid / not supported.

**These 4 errors should not occur, but are defined in case that the peripherals' firmware updates and its instructions change. **

- **InvalidCharacteristic:** Couldn't find the desired characteristic
- **WriteNotSupported:** When write operation is not supported by the characteristic.
- **ReadNotSupported:** When read operation is not supported by the characteristic.
- **SubscribeNotSupported:** When notify/indicate operation is not supported by the characteristic.

4.3. Range

This model is used to store data in format of a range with minimum, maximum and average value.

Sample usage:

```
new Range<int> {
    Min = 93,
    Average = 95,
    Max = 98
}
```

This model has been used in "OximeterMeasurement" to store the blood oxygen and pulse rate

4.4. Timestamp

This model defines a standard timestamp used to store fate and time received from the devices.

Different fields:

- **Year:** An int that represents the year value. Ranges from 0 to 9999.
- **Month:** An int that represents the month value. Ranges from 1 to 12.
- **Day:** An int that represents the day value. Ranges from 1 to 31.
- **Hour:** An int that represents the hour value. Ranges from 0 to 23.
- **Minute:** An int that represents the minutes' value. Ranges from 0 to 59.
- **Second:** An int that represents the seconds value. Ranges from 0 to 59.

Sample usage:

```
new Timestamp {
      Year = 2022,
      Month = 8,
      Day = 30,
      Hour = 15,
      Minute = 13,
      Second = 37
}
```

* This model has been used in all of the measurements received by peripherals *

4.5. Settings

This static class is used to hold the different values about our cli and load new values if needed.

Fields:

- ScanTimeout: Amount of seconds before
- AudioLenght: Length of the stethoscope recorded audio is seconds.
- UpdateWeight: A Boolean that determines if the user wants to see the live weight or not.
- Debug: A Boolean that is true when we want to debug our app and see all the logs.

4.6. GlucometerMeasurement

This model has been defined according to "Beurer GL50 evo" documentation and is used to parse the response bytes from peripheral to a valid measurement.

Fields:

- **Flags:** one byte that contains all the flags
- **SequenceNumber:** A 16-bit int that describes index of the current measurement inside peripherals memory.
- **Timestamp:** A variable of type Timestamp that stores date and time of the measurement.
- **Glucose:** The amount of blood glucose that has been measured by the peripheral. Value is stored in form on string. Example: “5.5 mmol/L”

4.7. OximeterMeasurement

This model has been defined according to "Beurer PO60" documentation and is used to parse the response bytes from peripheral to a valid measurement.

Fields:

- **Header:** one byte of header.
- **PacketNumber:** An 8-bit int that describes index of the current measurement inside the packet of 10 measurements.
- **Start:** A variable of type Timestamp that stores start date and time of the measurement.
- **End:** A variable of type Timestamp that stores end date and time of the measurement.
- **SpO2:** A variable of type Range<int> that stores minimum, average and maximum percentage of blood oxygen during the measurement.
- **PulseRate:** A variable of type Range<int> that stores minimum, average and maximum value of patients' pulse rate during the measurement.

4.8. ThermometerMeasurement

This model has been defined according to "Beurer FT95" documentation and is used to parse the response bytes from peripheral to a valid measurement.

Fields:

- **Flags:** a collection of flags as follows:
  - **Unit:** A string with value of "Celsius" or "Fahrenheit"
  - **TimestampFlag:** A Boolean that show the presence of the timestamp in the measurement.
  - **TypeFlag:** A Boolean that shows the presence of measurement type in the measurement.
  - **HasFever:** A Boolean that shows if patient has fever or not.
- **Temperature:** A float with 1 floating point that shows the temperature.
- **Timestamp:** A variable of type Timestamp that show the date and time of the measurement.
- **Type:** A string with value of "Body" if the thermometer measured body or "Object" if the thermometer measured an object.

4.9. BloodPressureMeasurement

This model has been defined according to "Beurer BM67" documentation and is used to parse the response bytes from peripheral to a valid measurement.

Fields:

- **Flags:** a collection of flags as follows:
  - **Unit:** A string with value of “mmHg” or “kpa”
  - **TimestampFlag:** A Boolean that show the presence of the timestamp in the measurement.
  - **PulseRateFlag:** A Boolean that shows the presence of pulse rate in the measurement.
  - **TypeFlag:** A Boolean that shows the presence of measurement type in the measurement.
  - **MeasurementStatusFlag:** A Boolean that shows the presence of measurement status in the measurement.
- **Systolic:** A float that shows the blood pressure systolic value.
- **Diastolic:** A float that shows the blood pressure diastolic value.
- **MeanArterialPressure:** A float that shows the patients mean arterial pressure. **Note that this value is always 0 in BM67 device. **
- **Timestamp:** A variable of type Timestamp that show the date and time of the measurement.
- **PulseRate:** A float that shows the patients pulse rate.
- **UserId:** An integer representing the patient id of the current measurement.
- **Status:** a collection of status’ as follows.
  - **BodyMovement:** A Boolean that determines if patient has moved during measurement or not.
  - **CuffFit:** A string that shows the cuff state with values of “Proper fit” or “Too loose”.
  - **IrregularPulse:** A Boolean that determines if patients pulse rate is irregular or not.
  - **IrregularRange:** A string that describes the pulse rate with values of “In Range”, “Exceeds Upper Limit” or “Below Lower Limit”.
  - **MeasurementPosition:** A string that shows the measurement position with values of “Proper Position” or “Improper Position”.

4.10. WeightMeasurement

This model has been defined manually and with the process of trial and error to extract data from it and is used to parse the response bytes from peripheral to a valid measurement.

Fields:

- **Unit:** A string that represents measurement unit. **This value is always “Kilograms”. **
- **Weight:** A float that represents patients weight with precision of 50 grams.

**Note that live weight measurement is 5 bytes long but the final result is longer. **

4.11. StethoscopeMeasurement

This model has been defined to store the location of the latest recorded audio and prints it to the user.

Fields:

- FileName: A string that describes the absolute path of the recorded audio file. E.g. “C:\\ProgramFiles\\Stethoscope\\Output.mp3”.

4.12. EchoMode

This model is an enumerated data type, used to describe the current recording mode of the stethoscope device.

Different values:

- **MODE_BELL_ECHO**
- **MODE_DIAFRAGM_ECHO**

5. Utils

This package contains all the utility files used inside the cli. In this section all the files inside "Utils" package are explained.

5.1. DataUtil

This file contains all the utilities used to convert different data types to each other.

List of functions:

- **Short2Bytes:** This function gets a 16-bit integer (short) and returns an array of length 2 that is equal to the original number.
- **byteArray2ShortArray:** This function gets an array of bytes (8-bit integers) and converts it to an array of shorts (16-bit integer) and returns it.
- **byteArray2ShortArray2:** This function does the same thing as the previous function. Only the conversion method is different (Previous: BitConverter, Current: Shift bits).

5.2. DecodeAudioData

This class is used to decode the audio data that has been received from the stethoscope.

List of functions:

- **decodeData:** This function receives an array of bytes and decodes its data using their encoding algorithm and returns the decoded data in form of a short array.

5.3. DecodeAudioDataUtils

This class is used to hold 2 instances of "DecodeAudioData", one for microphone data and one for speaker data. Also this class has an instance in form of a singleton that can be used from our code.

List of functions:

- **DecodeMicData:** Gets a byte array and returns the decoded data received from "decodeMicData" instance.
- **DecodeSpkData:** Gets a byte array and returns the decoded data received from "decodeSpkData" instance.

5.4. HandleRawData

This class does the most of the work by receiving the raw data, processing, decoding and clipping the distortion so that we can have a clear audio from the device.