Bluetooth LE Communication Protocol V4

The protocol uses one custom service which has one custom characteristic. The characteristic is configured to support `WriteWithoutResponse` and `Notify`. The phone sends data using `WriteWithoutResponse` and receives data via `Notify` indications. The device sends data using `Notify` and recieves data via `WriteWithoutResponse` indications.

The protocol assumes the application wants to exchange datagrams with a maximum length of 1500 bytes. When sending `<data>` it is first encoded as follows:

```
encoded = 0x00 || COBS-encode(<data> || CRC32C(<data>)) || 0x00
```

See:

- COBS - Constant Overhead Byte Stuffing
- CRC-32C

After encoding the sender breaks the `encoded` datagram into chunks with a maximum length of the current Bluetooth connection's `ATT MTU - 3` (default of 20 bytes). Then each chunk is submitted to the Bluetooth stack to be sent the other end.

The following python code shows an example encoding:

```
>>> import cobs
>>> import crc32c
>>> import struct
>>>
>>> data =
bytes.fromhex('a2478d771437863153ef00c3ce432be7a581cb78d94b5df6fbff28b8496c')
>>> crc32 = struct.pack('<I', crc32c.crc32c_update(0, data))
>>> encoded = b'\x00' + cobs.enc(data + crc32) + b'\x00'
>>>
>>> mtu = 20
>>> chunks = [ encoded[k:k+mtu] for k in range(0, len(encoded), mtu) ]
>>> for k, chunk in enumerate(chunks):
...     print(f'{k}: {chunk.hex()}')
...
0: 000ba2478d771437863153ef18c3ce432be7a581
1: cb78d94b5df6fbff28b8496cc97ec09f00
```

> NOTE: Bluetooth stacks have queues that will accept multiple `WriteWithoutResponse` or `Notify` requests, typically returning an error code with the queue fills, and a notification to the applicaiton that the queue one again has space. Bluetooth stacks also typically provide APIs to query the current ATT MTU or maximum length that can be sent via a single `WriteWithoutResponse` or `Notify`.

Command/Response Structure V1

The protocol is setup so that the phone is the initator and the device the responder. The following commands sent by the phone to the device:

- **Set Time** - allows the device to correct its internal timestamps with UTC provied by the phone.
- **Device Info** - queries details of the device, e.g. manufacturing date.
- **Config Get Ranges** - queries details of ranges of the configuration paramters.
- **Read Priority Data** - reads ACT metrics data.
- **Read Data** - reads compressed pressure data
- **Read Diagnostic Data** - read reset logs and BIST data.
- **Configure** - sets the configuration
- **Travel Mode** - places the device into travel mode.
- **FW Start Download** - prepare for a new FW image to be downloaded.
- **FW Next Block** - send the next block of the FW image.
- **FW Schedule Update** - schedules when to perform the FW update.

If there is a connection then the following events may be sent by the device to the phone:

- **New Data** - sent when new data (priority or normal or diagnostic) is available.
- **Device Status** - sent when the device status changes, e.g. battery level.

> NOTE: The `timestamp` fields reported by the device are guaranteed to be monotonically increasing.

> NOTE: The `seq` fields for **Read Priority Data**, **Read Data**, and **Read Diagnostic Data** are independent of each other.

> NOTE: the `seq` fields reported by the device are guaranteed to be monotonically increasing for the life of the device (as long as a **hard reset** has not been performed). Missing values may occur when the device resets while in the process of saving data to its local database, in which case the in-flight data is lost.

Commands, responses and events are decsribed by python dictionaries in the details below. When a phone or device is sending a command, response or event, it would encode the python dictionary using CBOR. The resulting byte stream would then be sent over Bluetooth using the **Bluetooth LE Communication Protocol** described above. For example:

```
>>> import cobs
>>> import crc32c
>>> import struct
>>> import cbor2
>>>
>>> cmd = {
...     'cmd': 'read-data',
...     'seq': 4096
... }
>>> data = cbor2.dumps(cmd)
>>> crc32 = struct.pack('<I', crc32c.crc32c_update(0, data))
>>> encoded = b'\x00' + cobs.enc(data + crc32) + b'\x00'
```

```
>>>
>>> mtu = 20
>>> chunks = [ encoded[k:k+mtu] for k in range(0, len(encoded), mtu) ]
>>> for k, chunk in enumerate(chunks):
...     print(f'{k}: {chunk.hex()}')
...
0: 0016a263636d6469726561642d64617461637365
1: 7119100567d63dbd00
```

Set Time

This command is sent by the phone each time the phone connects to the device. It should be the first message sent by the phone to the device after each new Bluetooth connection. It is used by the device to correct any offsets between the local timestamps (due to RTC clock drift) and the UTC time provided by the phone.

> NOTE: Errors in the provided `timestamp` will result in errors in the reported `timestamp` fields.

```
>>> import time
>>> cmd = {
...     'cmd': 'set-time',
...     'timestamp': int(time.time()+.5),       # <uint64> Phone UNIX epoc
timestamp in seconds
... }
>>> rsp = {
...     'rsp': 'set-time'
... }
```

Device Info

This command is typically sent by the phone each time the phone connects to the device. The values reported in response to this command do not change during a connection.

```
>>> import time
>>> cmd = {
...     'cmd': 'device-info',
... }
>>> rsp = {
...     'rsp': 'device-info',
...     'manu-date': int(time.time()+.5),  # <uint64> UNIX epoc timestamp in
seconds
...     'device-id': b'\x34\xa3\xcd\....', # <byte array> of device serial/id
...     'bl-ver': '1.0.1-0-g1a45f3e',      # boot loader version
...     'fw-ver': '2.3.0-0-gc31b4f0',      # main firmware image version
...     'ble-bonds': [3702043617, 2623609669], # <uint32 array> List of phones
that have paired with this device, as "Device IDs"
...                                        # TBD on more values
... }
```

Config Get Ranges

This command is typically sent by the phone each time the phone connects to the device. The values reported in response to this command do not change. Each entry specifies the [min, max] range (inclusive) in units that match the corresponding entries for the **Configuration** command.

```
>>> cmd = {
...     'cmd': 'config-get-ranges',
... }
>>> rsp = {
...     'rsp': 'config-get-ranges',
...     'led-toggle': [False, True]             # [<bool>, <bool>] min/max
...     'led-intensity': [0, 255],              # [<uint32>, <uint32>] min/max
...     'good-zone-threshold': [1.0, 15.0],     # [<float32>, <float23>] min/max
...     'high-zone-threshold': [10.0, 25.0],    # [<float32>, <float23>] min/max
...     'target-treatment-time': [60.0, 1200.0], # [<float32>, <float23>] min/max
...     'target-exhalation-count': [10, 40],    # [<uint32>, <uint32>] min/max
...     'minimum-breath-length': [5.0, 10.0],   # [<float32>, <float23>] min/max
...     'use-exhalation-count': [False, True]   # [<bool>, <bool>] min/max
... }
```

Read Priority Data

This command is typically sent when a **New Data** event is received and the reported `seq` by the device is different than the last `seq` received by the phone. The command is re-issued until there is no more data available. The `seq` in the command is used to indicate to the device that the phone has succesfully saved the read data with `seq`, so the corresponding item on the device can be deleted. There are various values for the `type` field in the response, each one has a different list of fields.

```
>>> import time
>>> cmd = {
...     'cmd': 'read-prio-data',
...     'seq': 35                                # <uint64> last item phone
successfully saved
... }
>>> # When there is data beyond item 35 and type is 'ACT'
>>> rsp = {
...     'rsp': 'read-prio-data',
...     'seq': 37,                               # <uint64> sequence number of this
item
...     'type': 'ACT',
...     'session': 123,                          # <uint64> Session id
...     'timestamp': int(time.time()+.5),        # <uint64> UNIX epoc timestamp in
seconds
...     'freq_avg': 18.0,                        # <float32> Average pulse
frequency over all breaths in Hz
...     'amp_avg': 17.7,                         # <float32> Average amplitude over
all breaths in cm-H2O
...     'peak_amp_avg': 27.2,                    # <float32> Average of peak
amplitude for each breath in cm-H2O
...     'peak_amp_std': 5.3,                     # <float32> Standard deviation of
peak amplitude for each breath in cm-H2O
...     'ppamp_avg': 3.7,                        # <float32> Average p-p pulse
amplitude over all breaths in cm-H2O
...     'duration_avg': 3.7,                     # <float32> Average breath
duration over all breath in seconds
...     'breath_count': 2,                       # <uint32> Total breath count
...     'duration': 7.6,                         # <float32> Total duration over
all breaths in seconds
...     'tppi': 2872.3,                          # <float32> TPPI metric over all
breaths in cm-H2O Hz s
...     'oc': 132.8,                             # <float32> Total pulse cycles
over all app breaths
...     'zone_percent': [float("NaN"), 0.152, 0.098, 0.033, 0.022, 0.043, 0.674] #
<float32 array> zone percent over all breaths [0, 1]
... }
>>> # When there is no more data beyond item 35 at this time
>>> rsp = {
...     'rsp': 'read-prio-data'
...}
```

Read Data

This command is typically sent when a **New Data** event is received and the reported `seq` by the device is different than the last `seq` received by the phone. The command is re-issued until there is no more data available. The `seq` in the command is used to indicate to the device that the phone has succesfully saved the read data with `seq`, so the corresponding item on the device can be deleted. There are various values for the `type` field in the response, each one has a different list of fields.

```
>>> import time
>>> cmd = {
...     'cmd': 'read-data',
...     'seq': 89                                # <uint64> last item phone
successfully saved
... }
>>> # When there is data beyond item 89
>>> rsp = {
...     'rsp': 'read-data',
...     'seq': 90,                               # <uint64> sequence number of this
item
...     'type': 'Pressure',
...     'data': b'\x27\x45.....',                # <byte array> variable length
compressed data
... }
>>> # Data could be "Start set"
>>> rsp = {
...     'rsp': 'read-data',
...     'seq': 90,                               # <uint64> sequence number of this
item
...     'type': 'Start set',
...     'session': 909,                        # <uint64> Session id
...     'set': 0,                              # <uint64> Set id. Restarts from 0
for each session
...     'timestamp': int(time.time()+.5),      # <uint64> UNIX epoc timestamp in
seconds
... }
>>> # or "End set"
>>> rsp = {
...     'rsp': 'read-data',
...     'seq': 90,                             # <uint64> sequence number of this
item
...     'type': 'End set',
...     'session': 909,                        # <uint64> Session id
...     'set': 0,                              # <uint64> Set id. Restarts from 0
for each session
...     'timestamp': int(time.time()+.5),      # <uint64> UNIX epoc timestamp in
seconds
...     'n_samples': 13056,                    # <uint32> Total number of samples
in compressed stream
...     'peak_error': 0.24,                    # <float32> Estimated peak
reconstructed error for stream
...     'rms_error': 0.01,                     # <float32> Estimated RMS
reconstructed error for stream
...     'compression_ratio': 0.03,             # <float32> Estimated compression
ratio for stream
... }
>>> # When there is no more data beyond item 89 at this time
>>> sp = {
...     'rsp': 'read-data'
...}
```

Read Diagnostic Data

This command is typically sent when a **New Data** event is received and the reported `seq` by the device is different than the last `seq` received by the phone. The command is re-issued until there is no more data available. The `seq` in the command is used to indicate to the device that the phone has succesfully saved the read data with `seq`, so the corresponding item on the device can be deleted. There are various values for the `type` field in the response, each one has a different list of fields.

```
>>> import time
>>> cmd = {
...     'cmd': 'read-diag-data',
...     'seq': 89                              # <uint64> last item phone
successfully saved
... }
>>> # When there is data beyond item 89
>>> rsp = {
...     'rsp': 'read-diag-data',
...     'seq': 90,                             # <uint64> sequence number of this
item
...     'type': 'Reset',
...     'seg': 0,                              # segment number
...     'count': 10,                           # total number of segments for
this reset
...     'timestamp': int(time.time()+.5),      # <uint64> UNIX epoc timestamp in
seconds
...     'data': b'\x27\x45.....',              # <byte array> variable length
compressed data
... }
>>> # When there is no more data beyond item 89 at this time
>>> sp = {
...     'rsp': 'read-diag-data'
...}
```

Configure

This command is typically sent on startup or when user has modified the configuration. The response contains the 'reflected' current settings. If there are any parameters in the command that are out of range, the corresponding setting is ignored (preserving the existing setting) and non-zero status is returned. If there are any parameters in the command that are missing, the device will report the current settings in the response.

```
>>> cmd = {
...     'cmd': 'configure',
...     'led-toggle': True,                   # <boolean> True to enable LED
feedback, False to disable
...     'led-intensity': 1,                   # <uint32> 0 for Low brightness, 1
for Med, 2 for High
...     'good-zone-threshold': 10.0,          # <float32> good zone threshold in
cm-H2O
...     'high-zone-threshold': 20.0,          # <float32> high zone threshold in
cm-H2O
...     'minimum-breath-length': 1.0,         # <float32> minimum breath length
in seconds
... }
>>> rsp = {
...     'rsp': 'configure',
...     'status': 0,                          # <int32> 0 if no error, otherwise
error code
...     'led-toggle': True,                   # <boolean> True to enable LED
feedback, False to disable
...     'led-intensity': 1,                   # <uint32> 0 for Low brightness, 1
for Med, 2 for High
...     'good-zone-threshold': 10.0,          # <float32> good zone threshold in
cm-H2O
...     'high-zone-threshold': 20.0,          # <float32> high zone threshold in
cm-H2O
...     'target-treatment-time': 270.0,       # <float32> target treatment time
in seconds
...     'target-exhalation-count': 20,        # <uint32> target exhalation count
...     'minimum-breath-length': 1.0,         # <float32> minimum breath length
in seconds
...      'use-exhalation-count': False,          # <boolean> Use exhalation count
(True) or use target treatment time (False)
... }
```

Erase

This command is sent to erase data that has been stored on the device, and/or to reset user settings to default. For the data partitions: 'delete' will just delete the records that are stored, and will not reset the sequence numbers of that partition. 'format' will do a flash erase of the entire partition, and will reset the next sequence number that will be stored back to 0. Formatting can take several seconds depending on the size of the partition.

```
>>> cmd = {
...      'cmd': 'erase',
...      'data': 'delete',                     # 'delete' or 'format' data
partition
...      'prio-data': 'format',                # 'delete' or 'format' priority data
partition
...      'diag-data': 'delete',                # 'delete' or 'format' diagnostic
data partition
...      'configuration': True,                # <boolean> Reset configuration to
default
...      'pairing': True,                      # <boolean> Erase BLE Pairing keys
... }
>>> rsp = {
...      'rsp': 'erase',
... }
>>> # Fields can be excluded if that type should not be erased
>>> # For example, the following would only reset configuration to default. All
other data would not be affected:
>>> cmd = {
...      'cmd': 'erase',
...      'configuration': True,                # <boolean> Reset configuration to
default
... }
```

Travel Mode

This command is sent when the user selects travel mode from the UI. After succesfully sending the response the device will close the connection and enter travel mode.

```
>>> cmd = {
...      'cmd': 'travel-mode',
... }
>>> rsp = {
...      'rsp': 'travel-mode',
... }
```

FW Start Download

This command is sent when the phone receives an hew FW image from the FW update server. It initializes the device to accept a new firmware image. The device may take up to 5 seconds before sending the response.

```
>>> cmd = {
...     'cmd': 'fw-start',
... }
>>> rsp = {
...     'rsp': 'fw-start',
...     'block-size': 256,          # <uint16> The maximum number of bytes in each
FW Next Block
... }
```

FW Next Block

After sending **FW Start Download** the phone should send as many **FW Next Block** commands as needed to complete the transfer of the FW image. Each block should of length `block-size` except the last, which may be less than `block-size`.

```
>>> cmd = {
...     'cmd': 'fw-block',
...     'data': b'\x23\x98\xfe....', # <byte array> max length of block-size.
... }
>>> rsp = {
...     'rsp': 'fw-block',
... }
```

FW Schedule Update

After transfering a FW image using **FW Start Download** and **FW Next Block** commands the phone can then schedule when to update the device FW with the new FW image using **FW Schedule Update**. If there is a Bluetooth connection at the scheduled update time, the device will disconnect before starting the update.

```
>>> cmd = {
...     'cmd': 'fw-schedule',
...     'timestamp': int(time.time()+.5), # <uint64> UNIX epoc timestamp in
seconds to update FW
... }
>>> # If scheduled
>>> rsp = {
...     'rsp': 'fw-schedule',
... }
>>> # If error, in which case FW update will not occur
>>> rsp = {
...     'rsp': 'fw-schedule',
...     'error': 'invalid',
... }
```

> NOTE: An error response occurs if the FW image is invalid for some reason, e.g. signature does not match, no FW image was down loaded using **FW Start Download** and **FW Next Block**, or timestamp is in the past.

> NOTE: It is possible to resend a **FW Schedule Update** command to reschedule the FW update. The new `timestamp` will replace the old one.

New Data

This event is sent by the device immediately after a new Bluetooth connection starts or when new data (prioirty or normal or diagnostic) written to the device queue and the corresponding device queue was empty (phone had read all data). The phone can use the reported seq numbers to determine if there is new data for it to read.

```
>>> event = {
  ...     'event': 'new-data',
  ...     'prio-data-seq':  67,  # <uint64> latest available seq number
  ...     'data-seq': 93,        # <uint64> latest available seq number
  ...     'diag-data-seq': 104,  # <uint64> latest available seq number
  ... }
```

Device Status

This event is sent by the device immediately after a new Bluetooth connection starts or when any of the field values change. On new Bluetooth connection all field are present. Subsequent **Device Status** events on the same Bluetooth connection may only include fields whose values have changed.

```
>>> event = {
...     'event': 'device-status',
...     'charging-state': 'Charged',   # <String> 'Not charging', 'Charging - CC, 
'Charging - CV', or 'Charged'
...     'charging-source': 'Wired',    # <String> 'Not connected', 'Wired, or 
'Wireless'
...     'session-id': 45,       # <uint64> Present if current session active, 
otherwise not present
...                             # TBD more values
... }
```

Future Directions

There are some planned features for future releases that would include streaming live data from the device to the phone. To support this there would be new commands to start/stop streaming the data, and a new event to support transporting the data. The data format For example:

```
>>> cmd = {
...     'cmd': 'stream-start',
...     'id': 'pressure',
... }
>>> rsp = {
...     'rsp': 'stream-start',
...     'id': 'pressure',
...     # TBD details on stream - units/format/fields/sample-rate/...
... }

>>> cmd = {
...     'cmd': 'stream-stop',
...     'id': 'pressure',
... }
>>> rsp = {
...     'rsp': 'stream-stop',
...     'id': 'pressure',
... }

>>> event = {
...     'event': 'stream-data',
...     'id': 'pressure',
...     'seq': 93,          # <uint64> seq number of this data segment
...     'timestamp': int(time.time()+.5),  # <uint64> UNIX epoc timestamp in
seconds of first sample in data
...     'data': [1,2, 3.4, -7.8, ...], # variable length array of data values.
... }
```