CHAPTER 9

Communication

RANDALL MAAS

This chapter is about the communication system:

- Internal communication with the base-board, and internal peripherals
- Bluetooth LE: with the Cube, and with the application
- WiFi: with the cloud, and with the application
- Internal support

1. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE

A significant part of Vector's software is focused on communication.

- Internal IPC between processes
- Communication with local peripherals and the base-board processor
- Communication with external accessories and applications.

The communication stacks:

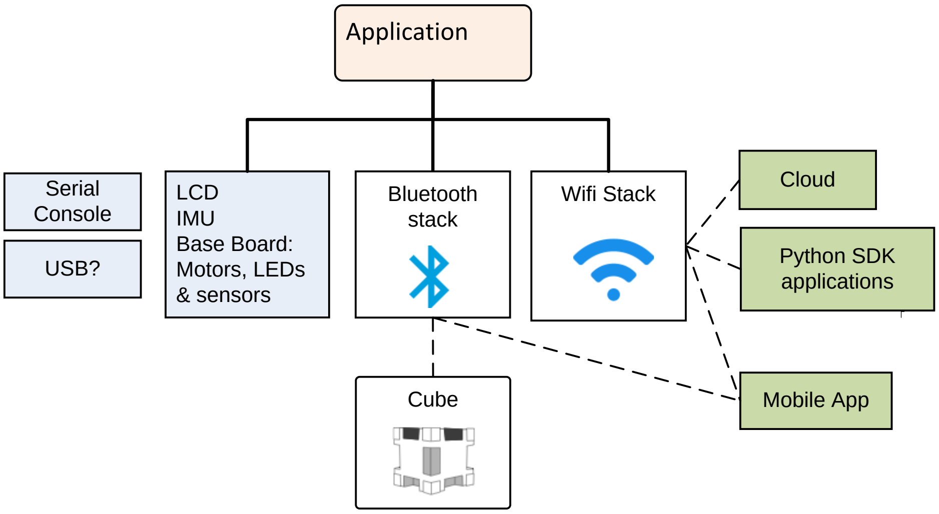


Figure 1: *The overall communication infrastructure*

2. INTERNAL COMMUNICATION WITH PERIPHERALS

Communication stack within the software. One part Linux, one part Qualcomm, and a big heaping dose of Anki's stuff.

2.1. COMMUNICATION WITH THE BASE-BOARD

The head board communicates with the base board using a serial interface. The device file is `/dev/ttyHS0`.

Data rate: 460800 bits/sec[1]

Messages from Base to Head are a regular, fixed-size packet, containing:

- The state of the backpack button
- The touch sensor voltage
- The microphone signals for all 4 microphones. (This could be 16 bits, or signed 8 bit for delta-sigma changes.)
- The battery voltage
- State of the charger (on dock/etc)
- The temperature of the battery or charger
- The state of 4 motor encoders, possibly as encoder counters, possibly as IO state
- The time of flight reading, probably 16bits in mm
- The voltage (or other signal) of each of the 4 cliff proximity sensors
- A CRC check

The messages from the head board to the base-board have the content:

- The 4 LED RGB states
- Controls for the motors: possible direction and enable; direction and duty cycle; or a target position and speed.
- Power control information: disable power to the system, turn off distance, cliff sensors, etc.

The messages are sent fast enough to support microphone sample rate of 15625 samples/second.

2.2. SERIAL BOOT CONSOLE

The head-board includes a 115200, 8 data bits no parity, 1 stop bit; the device file is `/dev/ttyHSL0`. Only prints the boot console. (This is passed in the commanded line by the bootloader)

2.3. USB

There are pins for USB on the head board. Asserting “F_USB” pad to VCC enables the port. During power-on, and initial boot it is a Qualcomm QDL port. The USB supports a Qualcomm debugging driver (QDL), but the readout is locked. It appears to be intended to inject firmware during manufacture.

The `/etc/initscriptsusb` file enables the USB and the usual functionfs adb. It lives in `/sbin/usr/composition/9091` (I think, if I understand the part number matching correctly). This launches ADB (DIAG + MODEM + QMI_RMNET + ADB)

Melanie T reports this not working, not enabled.

Vectors log shows the USB being disabled 24 seconds after linux starts.

[1] Value from the startup logs. Melanie T measured it on an oscilloscope and estimated it to be 2Mbps.

3. BLUETOOTH LE

Bluetooth LE is used for two purposes:

1. Bluetooth LE is used to initially configure Vector, to reconfigure him when the Wifi changes; to allow him to interact with the cube. Potentially allows some diagnostic and customization.
2. Bluetooth LE is used to communicate with the companion Cube accessory: to detect its movement, taps, and to set the state of its LEDs.

Vector's Bluetooth LE stack looks like:

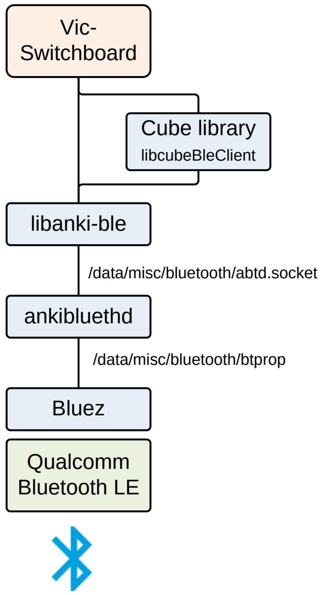


Figure 2: *The Bluetooth LE stack*

The elements of the Bluetooth LE stack include:

Table 1: *Elements of the Bluetooth LE stack*

Element	Description & Notes
ankibluetoothd	A server daemon. The application layer communicates with it over a socket; `/data/misc/bluetooth/abtd.socket`
BlueZ	Linux's official Bluetooth stack, including Bluetooth LE support. The Anki Bluetooth daemon interacts with it over a socket: /data/misc/bluetooth/btprop
bccmd	A Bluetooth core command
btmon	A command-line Bluetooth tool
libanki-ble.so	Communicates with Anki Bluetooth daemon probably serves both the external mobile application interface and communication with the companion cube.
libcubeBleClient.so[2]	A library to communicate with the companion cube, play animations on its LEDs, detect taps and the orientation of the cube.
viccubetool	Probably used to update the firmware in the Cube.

[2] The library includes great deal of built in knowledge of the state of application ("game engine"), animations, and other elements

4. COMMUNICATING WITH MOBILE APP AND SDK

Vector's *robot name* is something that looks like "Vector-E5S6". This name is used consistently; it will be Vector's:

- advertised Bluetooth LE peripheral name (although spaces are used instead of dashes)
- mDNS network name (dashes are used instead of spaces),
- the name used to sign certificates, and
- it will be the name of his WiFi Access Point, when placed into Access Point mode

4.1. CERTIFICATE BASED AUTHENTICATION

A *session token* is always provided by Anki servers.[3] It is passed to Vector to authenticate with him and create a client token. The session token is passed to Vector via the Bluetooth LE RTS protocol or the HTTPS-based SDK protocol; Vector will return a client token. The session token is single use only.

A *client token* is passed to Vector in each of the HTTPS-based SDK commands, and in the Bluetooth LE SDK Proxy commands. It is generated in one of two ways. One method is by the Bluetooth LE command (cloud session); the other is by posting "/v1/user_authentication" SDK request. The client token should be saved indefinitely for future use. It is not clear if the client token can be shared between the two transport mechanisms.

A *certificate* is also generated by Vector in the case of the SDK request. The certificate is intended to be added to the trusted SSL certificates before an HTTPS communication session. The certificate issued by Vector is good for 100 years.

The typical information embedded in a Vector certificate:

***Table 2:** Elements of a Vector certificate*

Element	Value
Common Name	*Vector's robot name*
Subject Alternative Names	*Vector's robot name*
Organization	Anki
Locality	SF
State	California
Country	US
Valid From	*the date the certificate was created*
Valid To	*100 years after the date the certificate was created*
Issuer	*Vector's robot name*, Anki
Serial Number	

[3] https://groups.google.com/forum/#!msg/anki-vector-rooting/YlYQsX08OD4/fvkAOZ91CgAJ
https://groups.google.com/forum/#!msg/anki-vector-rooting/XAaBE6e94ek/OdES50PaBQAJ

CHAPTER 10

Bluetooth LE Communication Protocol

This chapter is describes Vector's Bluetooth LE communication protocol.

- The kinds of activities that can be done thru communication channels
- The interaction sequences
- The communication protocol stack, including encryption, fragmentation and reassembly.

Note: communication with the Cube is simple reading and writing a characteristic, and covered in Appendix E.

5. COMMUNICATION PROTOCOL OVERVIEW

Vector advertises services on Bluetooth LE, with the Bluetooth LE peripheral name the same as his robot name (i.e. something that looks like "Vector-E5S6".)

Communication with Vector, once established, is structure as a request-response protocol. The request and responses are referred to as "C-Like Abstract Data structures" (CLAD) which are fields and values in a defined format, and interpretation. Several of these messages are used to maintain the link, setting up an encryption over the channel.

The application layer messages may be arbitrarily large. To support Bluetooth LE 4.1 (the version in Vector, and many mobile devices) the CLAD message must be broken up into small chunks to be sent, and then reassembled on receipt.

Combined with application-level encryption, the communication stack looks like:

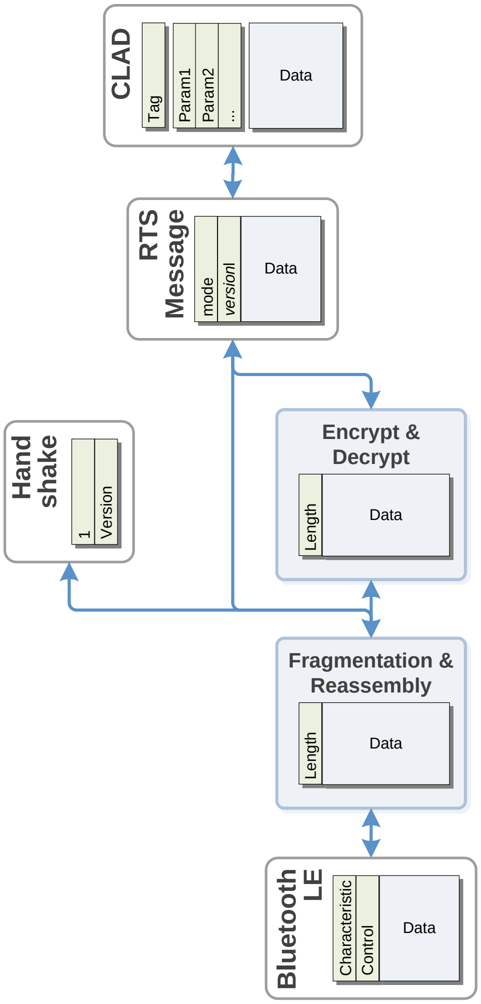


Figure 3: *Overview of encryption and fragmentation stack*

The Bluetooth LE is the link/transport media. It handles the delivery, and low-level error detection of exchanging message frames. The frames are fragments of the overall message. The GUID’s for the services and characteristics can be found in Appendix E.

The Fragmentation & Reassembly is responsible for breaking up a message into multiple frames, and reassembling them into a message.

The Encryption & Decryption layer is used to encrypt and decrypt the messages, after the communication channel has been set up.

The RTS is extra framing information that identifies the kind of CLAD message, and the version of its format. The format changed with version, so this version code is embedded at this layer.

The C-Like Abstract Data (CLAD) is the layer that decodes the messages into values for fields, and interprets them,

5.1. SETTING UP THE COMMUNICATION CHANNEL

It sometimes helps to start with the over all process. This section will walk thru the process, referring to later sections where detailed information resides.

If you use "first time" – or wish to re-pair with him – put him on the charger, and press the backpack button twice quickly. He'll display a screen indicating he is getting ready to pair.

If you have already paired the application with Vector, the encryption keys can be reused.

The process to set up a Bluetooth LE communication with Vector is complex. The sequence has many steps:

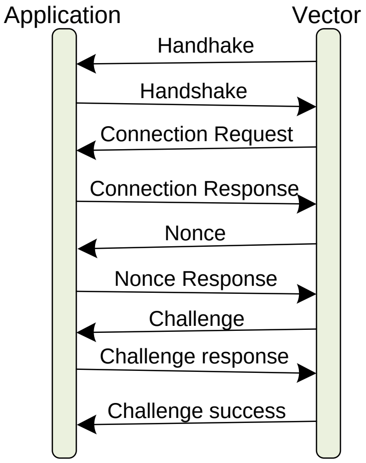


Figure 4: *Sequence for initiating communication with Vector*

1. The application opens Bluetooth LE connection (retrieving the service and characteristics handles), and subscribes to the "read" characteristic (see Appendix E for the UUID).
2. Vector sends *handshake* message; which the application receives. The handshake message structure is given below. The handshake message includes the version of the protocol supported.

Table 3: *Parameters for Handshake message*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*type*	?
1	4	uint32_t	*version*	The version of the protocol/messages to employ

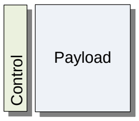
3. The application sends the handshake back
4. Then the Vector will send a *connection request*, consisting of the public key to use for the session. The application's response depends on whether this is a first time pairing, or a reuse.
    a. First time pairing requires that Vector have already been placed into pairing mode prior to connecting to Vector. The application keys should be created (see section *5.3.1 First time pairing* above).
    b. Reconnection can reuse the public and secret keys, and the encryption and decryption keys from a prior pairing
5. The application should then send the publicKey in the response

6. If this is a first time pairing, Vector will display a *pin code*. This is used to create the public and secret keys, and the encryption and decryption keys (see section *5.3.1 First time pairing* above). These can be saved for use in future reconnection.
7. Vector will send a *nonce* message. After the application has sent its response, the channel will now be encrypted.
8. Vector will send a *challenge* message. The application should increment the passed value and send it back as a challenge message.
9. Vector will send a *challenge success* message.
10. The application can now send other commands

If the user puts Vector on the charger, and double clicks the backpack button, Vector will usually send a *disconnect* request.

5.2. FRAGMENTATION AND REASSEMBLY

An individual frame sent over Bluetooth LE is limited to 20 bytes. (This preserves compatibility with Bluetooth LE 4.1) A frame looks like:



The control byte is used to tell the receiver how to *reassemble* the message using this frame.

- If the MSB bit (bit 7) is set, this is the start of a new message. The previous message should be discarded.
- If the 2nd MSB (bit 6) is set, this is the end of the message; there are no more frames.
- The 6 LSB bits (bits 0..5) are the number of payload bytes in the frame to use.

The receiver would append the payload onto the end of the message buffer. If there are no more frames to be received it will pass the buffer (and size count) on to the next stage. If encryption has been set up, the message buffer will be decrypted and then passed to the RTS and CLAD. If encryption has not been set up, it is passed directly to the RTS & CLAD.

Fragmenting reverses the process:

1. Set the MSB bit of the control byte, since this is the start of a message.
2. Copy up to 19 bytes to the payload.
3. Set the number of bytes in the 6 LSB bits of the control byte
4. If there are no more bytes remaining, set the 2nd MSB it of the control byte.
5. Send the frame to Vector
6. If there are bytes remaining, repeat from step 2.

5.3. ENCRYPTION SUPPORT

For the security layer, you will need the following:

***Example 1:** Bluetooth LE encryption structures*

```
uint8_t Vectors_publicKey[32];
uint8_t publicKey [crypto_kx_PUBLICKEYBYTES];
uint8_t secretKey [crypto_kx_SECRETKEYBYTES];
uint8_t encryptionKey[crypto_kx_SESSIONKEYBYTES];
uint8_t decryptionKey[crypto_kx_SESSIONKEYBYTES];
uint8_t encryptionNonce[24];
uint8_t decryptionNonce[24];
uint8_t pinCode[16];
```

The variables mean:

***Table 4:** The encryption variables*

Variable	Description
decryptionKey	The key used to decrypt each message from to Vector.
decryptionNonce	An extra bit that is added to each message. The initial nonce’s to use are provided by Vector.
encryptionKey	The key used to encrypt each message sent to Vector.
encryptionNonce	An extra bit that is added to each message as it is encrypted. The initial nonce’s to use are provided by Vector.
pinCode	6 digits that are displayed by Vector during an initial pairing.
Vectors_publicKey	The public key provided by Vector, used to create the encryption and decryption keys.

There are two different paths to setting up the encryption keys:

- First time pairing, and
- Reconnection

5.3.1 First time pairing

First time pairing requires that Vector be placed into pairing mode prior to the start of communication. This is done by placing Vector on the charger, and quickly double clicking the backpack button.

The application should generate its own internal *public* and *secret keys* at start.

***Example 2:** Bluetooth LE key pair*

```
crypto_kx_keypair(publicKey, secretKey);
```

The application will send a *connection response* with first-time-pairing set, and the public key. After Vector receives the connection response, he will display the *pin code*. (See the steps in the next section for when this will occur.)

The session *encryption* and *decryption keys* can then created:

***Example 3:** Bluetooth LE encryption & decryption keys*

```
crypto_kx_client_session_keys(decryptionKey, encryptionKey, publicKey, secretKey,
     Vector_publicKey);
size_t pin_length = strlen(pin);

crypto_generichash(encryptionKey, sizeof(encryptionKey), encryptionKey,
     sizeof(encryptionKey), pin, pin_length);
crypto_generichash(decryptionKey, sizeof(decryptionKey), decryptionKey,
     sizeof(decryptionKey), pin, pin_length);
```

5.3.2 Reconnecting

Reconnecting can reused the public and secret keys, and the encryption and decryption keys. It is not known how long these persist on Vector. {Next pairing? Next reboot? Indefinitely?}

5.3.3 Encrypting and decryption messages

Vector will send a *nonce* message with the *encryption* and *decryption nonces* to employ in encrypting and decrypting message.

Each received enciphered message can be decrypted from cipher text (cipher, and cipherLen) to the message buffer (message and messageLen) for further processing:

```
crypto_aead_xchacha20poly1305_ietf_decrypt(message, &messageLen, NULL, cipher,
      cipherLen,  NULL, 0L, decryptionNonce, decryptionKey);
sodium_increment(decryptionNonce, sizeof decryptionNonce);
```

***Example 4:** Decrypting a Bluetooth LE message*

Note: the decryptionNonce is incremented each time a message is decrypted.

Each message to be sent can be encrypted from message buffer (message and messageLen) into cipher text (cipher, and cipherLen) that can be fragmented and sent:

```
crypto_aead_xchacha20poly1305_ietf_encrypt(cipher, &cipherLen, message,
     messageLen, NULL, 0L, NULL, encryptionNonce, encryptionKey);
sodium_increment(encryptionNonce, sizeof encryptionNonce);
```

***Example 5:** Encrypting a Bluetooth LE message*

Note: the encryptionNonce is incremented each time a message is encrypted.

5.4. THE RTS LAYER

There is an extra, pragmatic layer before the messages can be interpreted by the application. The message has two to three bytes at the header:

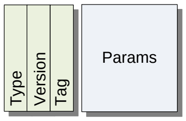


***Figure 5:** The format of an RTS frame*

- The type byte is either 1 or 4. If it is 1 the version of the message format is 1.
- If type byte is 4, the version is held in the next byte. (If the type is 1, there is no version byte).
- The next byte is the tag – the value used to interpret the message.

The tag, parameter body, and version are passed to the CLAD layer for interpretation. This is described in the next section.

5.5. FETCHING A LOG

The process to set up a Bluetooth LE communication with Vector is complex. The sequence has many steps:

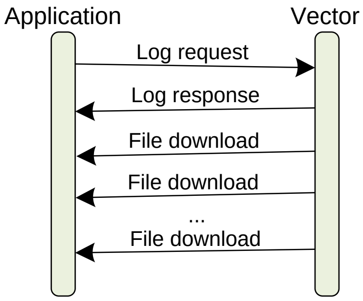


Figure 6: *Sequence for initiating communication with Vector*

The log request is sent to Vector. In principal this includes a list of the kinds of logs (called filter names) to be included. In practice, the "filter name" makes no difference.

Vector response, and if there will be a file sent, includes an affirmative and a 32-bit file identifier used for the file transfer.

Vector zips the log files up (as a tar.bz2 compressed archive) and sends the chunks to the application. Each chunk has this file identifier. (Conceptually there could be several files in transfer at a time.)

The file transfer is complete when the packet number matches the packet total.

6. MESSAGE FORMATS

This section describes the format and interpretation of the CLAD messages that go between the App and Vector. It describes the fields and how they are encoded, etc. Fields that do not have a fixed location, have no value for their offset. Some fields are only present in later versions of the protocol. They are marked with the version that they are present.

Except where otherwise stated:

- Requests are from the mobile application to Vector, and responses are Vector to the application
- All values in little endian order

Table 5: *Summary of the commands*

	Request	Response	Min Version
Application connection id	$1F_{16}$	20_{16}	4
Cancel pairing	10_{16}		0
Challenge	04_{16}	04_{16}	0
Challenge success	05_{16}		0
Connect	01_{16}	02_{16}	0
Cloud session	$1D_{16}$	$1E_{16}$	3
Disconnect	11_{16}		0
File download		$1a_{16}$	2
Log	18_{16}	19_{16}	2
Nonce	03_{16}	12_{16}	
OTA cancel	17_{16}		2
OTA update	$0E_{16}$	$0F_{16}$	0
SDK proxy	22_{16}	23_{16}	5
Response		21_{16}	4
SSH	15_{16}	16_{16}	0
Status	$0A_{16}$	$0B_{16}$	0
WiFi access point	13_{16}	14_{16}	0
WiFi connect	06_{16}	07_{16}	0
WiFi forget	$1B_{16}$	$1C_{16}$	3
WiFi IP	08_{16}	09_{16}	0
WiFi scan	$0C_{16}$	$0D_{16}$	0

6.1. APPLICATION CONNECTION ID

?

6.1.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*name length*	The length of the application connection id; may be 0
2	*varies*	uint8_t[name length]	*name*	The application connection id

Table 6: *Parameters for Application Connection Id request*

6.1.2 Response

There is no response.

6.2. CANCEL PAIRING

Speculation: this is sent by the application to cancel the pairing process

6.2.1 Request

The command has no parameters.

6.2.2 Response

There is no response.

6.3. CHALLENGE

This is sent by Vector if he liked the response to a nonce message.

6.3.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	4	uint8_t	*value*	The challenge value

Table 7: *Parameters for challenge request*

The application, when it receives this message, should increment the value and send the response (a challenge message).

6.3.2 Response

The parameters of the response body are:

Offset	Size	Type	Parameter	Description
0	4	uint8_t	*value*	The challenge value; this is 1 + the value that was received.

Table 8: *Parameters for challenge response*

If Vector accepts the response, he will send a *challenge success.*

6.4. CHALLENGE SUCCESS

This is sent by Vector if the challenge response was accepted.

6.4.1 Request

The command has no parameters.

6.4.2 Response

There is no response.

6.5. CLOUD SESSION

This command is used to request a cloud session.

6.5.1 Command

The parameters of the request body are:

Table 9: *Parameters for Cloud Session request*

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*session token length*	The number of bytes in the session token; may be 0
2	varies	uint8_t	*session token*	The session token, as received from the cloud server.[4]
	1	uint8_t	*client name length*	The number of bytes in the client name string; may be 0 version >= 5
	varies	uint8_t[]	*client name*	The client name string. Informational only. The mobile app uses the name of the mobile device. version >= 5
	1	uint8_t	*application id length*	The number of bytes in the application id string; may be 0; version >= 5
	varies	uint8_t[]	*application id*	The application id. Informational only. The mobile uses “companion-app”. version >= 5

6.5.2 Response result

The parameters for the connection response message:

Table 10: *Parameters for Cloud Session Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*success*	0 if failed, otherwise successful
1	1	uint8_t	*status*	See *Table 11: Cloud status enumeration*
2	1	uint16_t	*client token GUID length*	The number of bytes in the client token GUID; may be 0
	varies	uint8_t[]	*client token GUID*	The client token GUID. The client token GUID should be saved for future use.

The cloud status types are:

Table 11: *Cloud status enumeration*

Index	Meaning
0	unknown error
1	connection error
2	wrong account
3	invalid session token
4	authorized as primary
5	authorized as secondary
6	reauthorization

[4] https://groups.google.com/forum/#!msg/anki-vector-rooting/YlYQsX08OD4/fvkAOZ91CgAJ
https://groups.google.com/forum/#!msg/anki-vector-rooting/XAaBE6e94ek/OdES50PaBQAJ

6.6. CONNECT

The connect request *comes from Vector* at the start of a connection. The response is from the application.

6.6.1 Request

The parameters of the request body are:

***Table 12:** Parameters for Connection request*

Offset	Size	Type	Parameter	Description
0	32	uint8_t[32]	*publicKey*	The public key for the connection

The application, when it receives this message, should use the public key for the session, and send a response back.

6.6.2 Response

The parameters for the connection response message:

***Table 13:** Parameters for Connection Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*connectionType*	See *Table 14: Connection types enumeration*
1	32	uint8_t[32]	*publicKey*	The public key to use for the connection

The connection types are:

***Table 14:** Connection types enumeration*

Index	Meaning
0	first time pairing (requests pin code to be displayed)
1	reconnection

The application sends the response, with its publicKey (see section *5.3 Encryption support*). A "first time pairing" connection type will cause Vector to display a pin code on the screen

If a first time pairing response is sent:

- If Vector is not in pairing mode – was not put on his charger and the backpack button pressed twice, quickly – Vector will respond. Attempting to enter pairing mode now will cause Vector to send a *disconnect* request.
- If Vector is in pairing mode, Vector will display a pin code on the screen, and send a nonce message, triggering the next steps of the conversation.

If a reconnection is sent, the application would employ the public and secret keys, and the encryption and decryption keys from a prior pairing.

6.7. DISCONNECT

This may be sent by Vector if there is an error, and it is ending communication. For instance, if Vector enters pairing mode, it will send a disconnect.

The application may send this to request Vector to close the connection.

6.7.1 Request

The command has no parameters.

6.7.2 Response

There is no response.

6.8. FILE DOWNLOAD

This command is used to pass chunks of a file to Vector. Files are broken up into chunks, and sent.

6.8.1 Request

There is no direct request.

6.8.2 Response

The parameters of the response body are:

Table 15: *Parameters for File Download request*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*status*	
1	4	uint32_t	*file id*	
5	4	uint32_t	*packet number*	The chunk within the download
9	4	uint32_t	*packet total*	The total number of packets to be sent for this file download
13	2	uint16_t	*length*	The number of bytes to follow (can be 0)
	varies	uint8_t[length]	*bytes*	The bytes of this file chunk

6.9. LOG

This command is used to request the Vector send a compressed archive of the logs.

6.9.1 Request

The parameters of the request body are:

Table 16: *Parameters for Log request*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*mode*	
1	2	uint16_t	*num filters*	The number of filters in the array
3	varies	filter[num filters]	*filters*	The filter names

Each filter entry has the following structure:

Table 17: *Log filter*

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*filter length*	The length of the filter name; may be 0
2	*varies*	uint8_t[filter length]	*filter name*	The filter name

6.9.2 Response

It can take several seconds for Vector to prepare the log archive file and send a response. The response will be a “log response” (below) and a series of “file download” responses.

The parameters for the response message:

Table 18: *Parameters for Log Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*exit code*	
1	4	uint32_t	*file id*	A 32-bit identifier that will be used in the file download messages.

6.10. NONCE

A nonce is sent by Vector after he has accepted your key, and the application sends a response

6.10.1 Request

The parameters for the nonce request message:

Offset	Size	Type	Parameter	Description
0	24	uint8_t[24]	*toVectorNonce*	The nonce to use for sending stuff to Vector
24	24	uint8_t[24]	*toAppNonce*	The nonce for receiving stuff from Vector

***Table 19:** Parameters for Nonce request*

6.10.2 Response

After receiving a nonce, if the application is in first-time pairing the application should send a response, with a value of 3.

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*connection tag*	This is always 3

***Table 20:** Parameters for Nonce response*

After the response has been sent, the channel will now be encrypted. If vector likes the response, he will send a challenge message.

6.11. OTA UPDATE

This command is used to request the Vector download firmware from a given server

6.11.1 Request

The parameters of the request body are:

Table 21: *Parameters for OTA request*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*length*	The length of the URL; may be 0
1	*varies*	uint8_t[length]	*URL*	The URL string

6.11.2 Response

The response will be one or more "OTA response" indicating the status of the update, or errors. Status codes >= 200 indicate that the update process has completed. The update has completed the download when the current number of bytes match the expected number of bytes.

The parameters for the response message:

Table 22: *Parameters for OTA Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*status*	See *Table 23: OTA status enumeration*
1	8	uint64_t	*current*	The number of bytes downloaded
9	8	uint64_t	*expected*	The number of bytes expected to be downloaded

The OTA status codes are:

Table 23: *OTA status enumeration*

Status	Meaning
0	idle
1	unknown
2	in progress
3	complete
4	rebooting
5	error
200…	Status codes from the update-engine. See Appendix C, *Table 44: OTA update-engine status codes* for these update-engine status codes.

Note: the status codes 200 and above are from the update-engine, and are given in Appendix C.

6.12. RESPONSE

This message will be sent on the event of an error. Primarily if the session is not cloud authorized and the command requires it.

Table 24: Parameters for Response

Offset	Size	Type	Parameter	Description
0	1	uint16_t	*code*	0 if not cloud authorized, otherwise authorized
1	1	uint8_t	*length*	The number of bytes in the string that follows.
	varies	uint8_t [length]	*text*	A text error message.

6.13. SDK PROXY

This command is used to pass the gRPC/protobufs messages to Vector over Bluetooth LE. It effectively wraps a HTTP request/response. Note: the HTTPS TLS certificate is not employed with this command.

6.13.1 Request

The parameters of the request body are:

Table 25: *Parameters for the SDK proxy request*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*GUID length*	The number of bytes in the GUID string; may be 0
2	*varies*	uint8_t[GUID length]	*GUID*	The GUID string
	1	uint8_t	*msg length*	The number of bytes in the message id string
	varies	uint8_t[msg id length]	*msg id*	The message id string
	1	uint8_t	*path length*	The number of bytes in the URL path string
	varies	uint8_t[path length]	*path*	The URL path string
	2	uint16_t	*JSON length*	The length of the JSON
	varies	uint8_t[JSON length]	*JSON*	The JSON (string)

6.13.2 Response

The parameters for the response message:

Table 26: *Parameters for the SDK proxy Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*msg id length*	The number of bytes in the message id string; may be 0
2	*varies*	uint8_t[msg id length]	*msg id*	The message id string
	2	uint16_t	*status code*	The HTTP-style status code that the SDK may return.
	1	uint8_t	*type length*	The number of bytes in the response type string
	varies	uint8_t[type length]	*type*	The response type string
	2	uint16_t	*body length*	The length of the response body
	varies	uint8_t[body length]	*body*	The response body (string)

6.14. SSH

This command is used to request the Vector allow SSH. It is reported that only the developer releases support SSH; it is not known which versions are applicable. It does not appear that SSH can be enabled in the release firmware.

6.14.1 Request

The parameters for the request message:

Table 27: *Parameters for SSH request*

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*num keys*	The number of SSH authorization keys; may be 0
2	*varies*	keys[num keys]	*keys*	The array of authorization key strings (see below).

Each authorization key has the following structure:

Table 28: *SSH authorization key*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*key length*	The length of the key; may be 0
1	*varies*	uint8_t[key length]	*key*	The SSH authorization key

6.14.2 Response

The response has no parameters.

6.15. STATUS

This command is used to request basic info from Vector.

6.15.1 Request

The request has no parameters.

6.15.2 Response

The parameters for the response message:

Table 29: *Parameters for Status Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*SSID length*	The number of bytes in the SSID string; may be 0
2	*varies*	uint8_t[SSID length]	*SSID*	The WiFi SSID (hex string).
	1	uint8_t	*WiFi state*	See *Table 30: WiFi state enumeration*
	1	uint8_t	*access point*	0 not acting as an access point, otherwise acting as an access point
	1	uint8_t	*Bluetooth LE state*	0 if the Bluetooth
	1	uint8_t	*Battery state*	
	1	uint8_t	*version length*	The number of bytes in the version string; may be 0 version >= 2
	varies	uint8_t [version length]	*version*	The version string; version >= 2
	1	uint8_t	*ESN length*	The number of bytes in the ESN string; may be 0 version >= 4
	varies	uint8_t[ESN length]	*ESN*	The *electronic serial number* string; version >= 4
	1	uint8_t	*OTA in progress*	0 over the air update not in progress, otherwise in process of over the air update; version >= 2
	1	uint8_t	*has owner*	0 does not have owner, otherwise has owner; version >= 3
	1	uint8_t	*cloud authorized*	0 is not cloud authorized, otherwise is cloud authorized; version >= 5

Note: a *hex string* is a series of bytes with values 0-15. Every pair of bytes must be converted to a single byte to get the characters. Even bytes are the high nibble, odd bytes are the low nibble.

The WiFi states are:

Table 30: *WiFi state enumeration*

Index	Meaning
0	Unknown
1	Online
2	Connected
3	Disconnected

6.16. WIFI ACCESS POINT

This command is used to request that the Vector act as a WiFi access point. This command requires that a "cloud session" have been successfully started first (see section *6.5 Cloud session*).

If successful, Vector will provide a WiFi Access Point with an SSID that matches his robot name.

6.16.1 Request

The parameters of the request body are:

Table 31: *Parameters for WiFi Access Point request*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*enable*	0 to disable the WiFi access point, 1 to enable it

6.16.2 Response

If the Bluetooth LE session is not cloud authorized a "response" message will be sent with this error. Otherwise the WiFi Access Point response message will be sent.

The parameters for the response message:

Table 32: *Parameters for WiFi Access Point Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*enabled*	0 if the WiFi access point is disabled, otherwise enabled
1	1	uint8_t	*SSID length*	The number of bytes in the SSID string; may be 0
2	*varies*	uint8_t[SSID length]	*SSID*	The WiFi SSID (hex string)
	1	uint8_t	*password length*	The number of bytes in the password string; may be 0
	varies	uint8_t [password length]	*password*	The WiFi password

6.17. WIFI CONNECT

This command is used to request Vector to connect to a given WiFi SSID. Vector will retain this WiFi for future use.

6.17.1 Request

The parameters for the request message:

Table 33: *Parameters for WiFi Connect request*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*SSID length*	The number of bytes in the SSID string; may be 0
1	*varies*	uint8_t[SSID length]	*SSID*	The WiFi SSID (hex string)
	1	uint8_t	*password length*	The number of bytes in the password string; may be 0
	varies	uint8_t [password length]	*password*	The WiFi password
	1	uint8_t	*timeout*	How long to given the connect attempt to succeed.
	1	uint8_t	*auth type*	The type of authentication to employ; see *Table 34: WiFi authentication types enumeration*
	1	uint8_t	*hidden*	0 the access point is not hidden; 1 it is hidden

The WiFi authentication types are:

Table 34: *WiFi authentication types enumeration*

Index	Meaning
0	None, open
1	WEP
2	WEP shared
3	IEEE8021X
4	WPA PSK
5	WPA2 PSK
6	WPA2 EAP

6.17.2 Response

The parameters for the response message:

Table 35: *Parameters for WiFi Connect command*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*SSID length*	The length of the SSID that was deleted; may be 0
1	*varies*	uint8_t[SSID length]	*SSID*	The SSID (hex string) that was deleted
	1	uint8_t	*WiFi state*	See *Table 30: WiFi state enumeration*
	1	uint8_t	*connect result*	version >= 3

6.18. WIFI FORGET

This command is used to request Vector to forget a WiFi SSID.

6.18.1 Request

The parameters for the request message:

Table 36: *Parameters for WiFi Forget request*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*delete all*	0 if Vector should delete only one SSID; otherwise Vector should delete all SSIDs
1	1	uint8_t	*SSID length*	The length of the SSID that to be deleted; may be 0
2	*varies*	uint8_t[SSID length]	*SSID*	The SSID (hex string) to be deleted

6.18.2 Response

The parameters for the response message:

Table 37: *Parameters for WiFi Forget response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*did delete all*	0 if only one; otherwise Vector deleted all SSIDs
1	1	uint8_t	*SSID length*	The length of the SSID that was deleted; may be 0
2	*varies*	uint8_t[SSID length]	*SSID*	The SSID (hex string) that was deleted

6.19. WIFI IP ADDRESS

This command is used to request Vector's WiFi IP address.

6.19.1 Request

The request has no parameters

6.19.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*has IPv4*	0 if Vector doesn't have an IPv4 address; other it does
1	1	uint8_t	*has IPv6*	0 if Vector doesn't have an IPv6 address; other it does
2	4	uint8_t[4]	*IPv4 address*	Vector's IPv4 address
6	32	uint8_t[16]	*IPv6 address*	Vector's IPv6 address

Table 38: *Parameters for WiFi IP Address response*

6.20. WIFI SCAN

This command is used to request Vector to scan for WiFi access points.

6.20.1 Request

The command has no parameters.

6.20.2 Response

The parameters for the response message:

Table 39: *Parameters for WiFi scan response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*status code*	
1	1	uint8_t	*num entries*	The number of access points in the array below
2	*varies*	AP[num entries]	*access points*	The array of access points

Each access point has the following structure:

Table 40: *Parameters access point structure*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*auth type*	The type of authentication to employ; see *Table 34: WiFi authentication types enumeration*
1	1	uint8_t	*signal strength*	The number of bars, 0..4
2	1	uint8_t	*SSID length*	The length of the SSID string
3	*varies*	uint8_t[SSID length]	*SSID*	The SSID (hex string)
	1	uint8_t	*hidden*	0 not hidden, 1 hidden; version >= 2
	1	uint8_t	*provisioned*	0 not provisioned, 1 provisioned; version>= 3

Appendices

- ABBREVIATIONS, ACRONYMS, & GLOSSARY. This appendix provides a gloss of terms, abbreviations, and acronyms.
- FAULT AND STATUS CODES. This appendix provides describes the system fault codes, and update status codes.
- BLUETOOTH LE PROTOCOLS. This appendix provides information on the Bluetooth LE interfaces to the companion Cube, and to Anki Vector.

[This page is intentionally left blank for purposes of double-sided printing]

APPENDIX A

Abbreviations, Acronyms, Glossary

Table 41: *Common acronyms and abbreviations*

Abbreviation / Acronym	Phrase
ADC	analog to digital converter
AG	animation group
AVS	Alexa Voice Service
BIN	binary file
CCIS	customer care information screen
CLAD	C-like abstract data structures
CRC	cyclic redundancy check
DAS	*unknown (diagnostic/data analytics service?)*
DFU	device firmware upgrade
EEPROM	electrical-erasable programmable read-only memory
EMR	*unknown (emergency mode recovery?)*
ESD	electro-static discharge
ESN	electronic serial number
FBS	flat buffers
FDE	full disc encryption
GPIO	general purpose IO
GUID	globally unique identifier (effectively same as UUID)
I2C	inter-IC communication
IMU	inertial measurement unit
IR	infrared
JDocs	JSON Documents
JSON	javascript object notion
JTAG	Joint Test Action Group
LCD	liquid crystal display
LED	light emitting diode
LUKS	linux unified key setup
MCU	microcontroller
mDNS	multicast domain name service (DNS)

MEMS	micro-electromechanical systems
MISO	master-in, slave-out
MOSI	master-out, slave-in
MPU	microprocessor
MSRP	manufacturer's suggest retail price
OLED	organic light-emitting diode display
OTA	over the air updates
PCB	printed circuit board
PCBA	printed circuit board assembly (PCB with the components attached)
PMIC	power management IC
PWM	pulse width modulation
QSN	Qualcomm serial number
RPM	resource power management
RRT	rapidly-expanding random tree
SCLK	(I2C) serial clock
SDA	(I2C) serial data
SDK	software development kit
SLAM	simultaneous localization and mapping
SPI	serial-peripheral interface
SSH	secure shell
SSID	service set identifier (the name of the Wifi network)
STM32	A microcontroller family from ST Microelectronics
SWD	single wire debug
TAR	tape archive file
TTS	text to speech
UART	universal asynchronous receiver/transmitter
USB	universal serial bus
UUID	universally unique identifier (effectively same as GUID)
vic	short Victor (Vector's working name)

Table 42: *Glossary of common terms and phrases*

Phrase	Description
A*	A path finding algorithm
aboot	The Android boot-loader used to launch Vector's linux system.
attitude	orientation
boot loader	A piece of software used to load and launch the application firmware.
C-like abstract data structure	Anki's phrase for when information is packed into fields and values with a defined binary format, and interpretation. (Protobufs are often used for the same purpose.)
capacitive touch	

certificate	Vector generates an SSL certificate that can be used for the secure communications.
characteristic (Bluetooth LE)	A key (or slot) that holds a value in the services key-value table. A characteristic is uniquely identified by its UUID.
client token	A string token provided by Vector that is passed with each SDK command.
control	motors and forces to move where and how it is told to. (smooth arcs)
D*-lite	A path-finding algorithm
device mapper verity (dm-verity)	A feature of the Linux kernel that checks the boot and RAM file systems for alteration, using signed keys
flash	A type of persistent (non-volatile) storage media.
guidance	Builds the desired path
navigation	Knowing where it is in the map
nonce	An initially random number, incremented after each use .
path planning	smooth arcs and line segments
pose	position and orientation of an object relative to a coordinate system
power source	Where the electric energy used to power Vector comes from.
rapidly-expanding random tree	A path-finding algorithm
recovery mode	
robot name	Vector robot name looks like “Vector-E5S6”. It is “Vector-“ followed by a 4 letters and numbers.
session token	A string token provided by the Anki servers that is passed to Vector to authenticate with him and create a *client token.*
simultaneous localization and mapping	A vision based technique for building a map of the immediate world for purposes of positioning oneself within it, and detecting relative movements.
service (Bluetooth LE)	A key-value table, grouped together for a common purpose. A service is uniquely identified by its UUID.
Trust Zone	A security mode on ARM processor where privileged/special code is run. This includes access to encryption/decryption keys.
universally unique identifier (UUID)	A 128bit number that is unique. (effectively same as GUID)

APPENDIX C

Fault and status codes

The following are system status codes that may be produced during startup:

Table 43: *The system fault codes*

Code	Meaning
1..10	Systemd failed…?
200…	Software update status code, see table below
700-705	Internal sensor out of range or failed
800	Vic-anim was unable to start or crashed
801	?
898	? “general hardware disconnect”
899	?
913	Vic-switchboard was unable to start or crashed
914	Vic-engine was unable to start or crashed
915	?
916	Vic-robot was unable to start or crashed
917	?
920	Vic-gateway-cert was unable to generate a x509 certificate for Vic-gateway
921	Vic-gateway was unable to start or crashed
923	Vic-cloud was unable to start or crashed
980	?

The following are the update-engine status codes that may be produced during the update process:

Table 44: *OTA update-engine status codes*

Status	Meaning
200	The TAR contents did not follow the expected order.
201	Unhandled section format for expansion, or The manifest version is not supported, or The OTA has the wrong number of images for the type, or The OTA is missing a BOOT or SYSTEM image, or The manifest configuration is not understood
202	Could not mark target, a, or b slot unbootable, or Could not set target slot as active
203	Unable to construct automatic update URL, or The URL could not be opened
204	The file wasn’t a valid TAR file, or is corrupt
205	The compression scheme is not supported, or Decompression failed, the file may be corrupt
207	Delta payload error
208	Couldn't sync OS images to disk, or Disk error while transferring OTA file.

209	The manifest failed signature validation; or the aboot, boot image, system image, or delta.bin hash doesn't match signed manifest
210	The encryption scheme is not supported.
211	Vector’s current version doesn’t match the baseline for a delta update.
212	The decompression engine had an unexpected, undefined error.
213	QSN doesn't match manifest
214	There is a mismatch: development Vectors can’t install release OTA firmware, and release Vectors can’t install development OTA firmware.
215	OTA transfer failed, due to timeout.
216	OS version name in the update file doesn’t follow an acceptable pattern, or it is not allowed to upgrade or downgrade from the current version to the new version.
219	Other unexpected, undefined error while transferring OTA file.

APPENDIX E

Bluetooth LE Services & Characteristics

This Appendix describes the configuration of the Bluetooth LE services – and the data access they provide – for the accessory cube and for Vector.

7. CUBE SERVICES

times and other feature parameters:

Table 45: The Bluetooth LE services

Service	UUID[5]	Description & Notes
Device Info Service[6]	$180A_{16}$	Provides device and unit specific info –it's manufacturer, model number, hardware and firmware versions
Generic Access Profile[7]	1800_{16}	The device name, and preferred connection parameters
Generic Attribute Transport[8]	1801_{16}	Provides access to the services.
Cube's Service	C6F6C70F-D219-598B-FB4C-308E1F22F830$_{16}$	Service custom to the cube, reporting battery, accelerometer and date of manufacture

Note: It appears that there isn't a battery service on the Cube. When in over-the-air update mode, there may be other services present (i.e. by a bootloader)

Table 46: The Cube's Device info settings

Element	Value
Device Name (Default)	"Vector Cube"
Firmware Revision	"v_5.0.4"
Manufacturer Name	"Anki"
Model Number	"Production"
Software Revision	"2.0.0"

[5] All values are a little endian, per the Bluetooth 4.0 GATT specification
[6] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.device_information.xml
[7] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_access.xml
[8] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_attribute.xml

7.1. CUBE'S ACCELEROMETER SERVICE

Values are little-endian, except where otherwise stated.

Table 47: *Cube's accelerometer service characteristics*

UUID	Access	Size	Notes
0EA75290-6759-A58D-7948-598C4E02D94A$_{16}$	*Write*	unknown	
450AA175-8D85-16A6-9148-D50E2EB7B79E$_{16}$	Read	The date and time of manufacture (?)	
		char[]	*A date and time string*
43EF14AF-5FB1-7B81-3647-2A9477824CAB$_{16}$	Read, Notify, Indicate	Reads the battery and acceleromter	
		uint16_t	*battery ADC value*
		uint16_t	*accelerometer X ADC value #1*
		uint16_t	*accelerometer Y ADC value #1*
		uint16_t	*accelerometer Z ADC value #1*
		uint16_t	*accelerometer X ADC value #2*
		uint16_t	*accelerometer Y ADC value #2*
		uint16_t	*accelerometer Z ADC value #2*
		uint16_t	*accelerometer X ADC value #3*
		uint16_t	*accelerometer Y ADC value #3*
		uint16_t	*accelerometer Z ADC value #3*
9590BA9C-5140-92B5-1844-5F9D681557A4$_{16}$	Write		Unknown

Presumably some of these will cause the Cube to go into over the air update (OTAU) mode, allowing its firmware to be updated.

Others turn the RGB on to an RGB color, possibly duty cycle and pulsing duty cycle

8. VECTOR SERVICES SERVICE

times and other feature parameters:

Table 48: *Vector's Bluetooth LE services*

Service	UUID[9]	Description & Notes
Generic Access Profile	1800$_{16}$	The device name, and preferred connection parameters
Generic Attribute Transport	1801$_{16}$	Provides access to the services.
Vector's Serial Service	FEE3$_{16}$	The service with which we can talk to Vector.

It appears that there isn't a battery service on the Vector.

Table 49: *The Vector's Device info settings*

Element	Value
Device Name (Default)	"Vector" followed by his serial number

[9] All values are a little endian, per the Bluetooth 4.0 GATT specification

8.1. VECTOR'S SERIAL SERVICE

UUID	Access	Format	Notes
30619F2D-0F54-41BD-A65A-7588D8C85B45$_{16}$	Read, Notify,Indicate		
7D2A4BDA-D29B-4152-B725-2491478C5CD7$_{16}$	write		

***Table 50:** Vector's serial service characteristics*