CHAPTER 7

Bluetooth LE Communication Protocol

RANDALL MAAS 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 C.

1. COMMUNICATION PROTOCOL OVERVIEW

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:

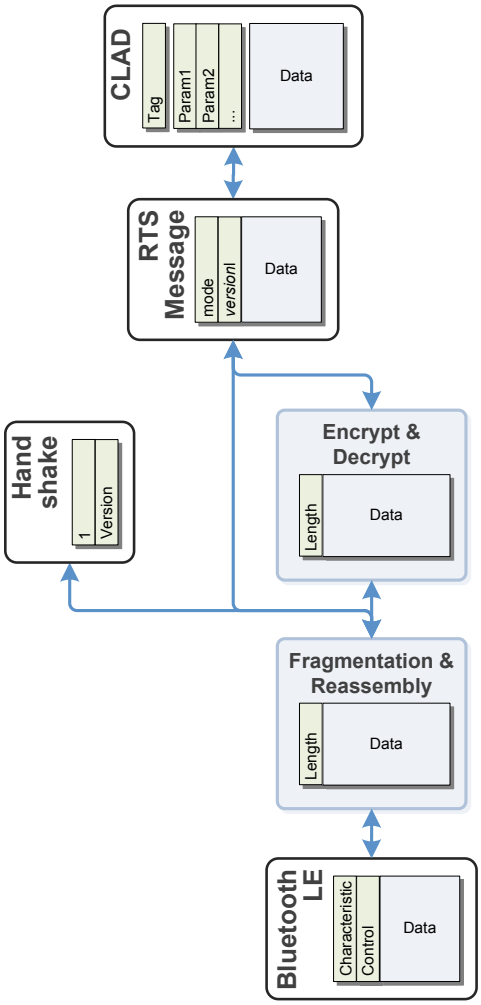

Figure 1: *Overview of encryption and fragmentation stack*

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 C.

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,

1.1. 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.

1. The application opens Bluetooth LE connection (retrieving the service and characteristics handles), and subscribes to the "read" characteristic (see Appendix C 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 1: *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 *1.3.1 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 *1.3.1 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.

1.2. 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:

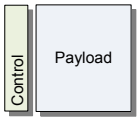

Figure 2: *The format of a frame*

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 byte remaining, repeat from step 2.

1.3. ENCRYPTION SUPPORT

For the security layer, you will need the following

```
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 2: *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

1.3.1 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.

```
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:

```
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);
```

1.3.2 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?}

1.3.3 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);
```

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);
```

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

1.4. 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:

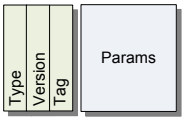

Figure 3: *The format of an RTS frame*

- 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.

2. 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 3: *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	26_{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

2.1. APPLICATION CONNECTION ID

?

2.1.1 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 4: *Parameters for Application Connection Id request*

2.1.2 Response

There is no response.

2.2. CANCEL PAIRING

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

2.2.1 Request

The command has no parameters.

2.2.2 Response

There is no response.

2.3. CHALLENGE

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

2.3.1 Request

The parameters of the request body are:

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

Table 5: *Parameters for challenge request*

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

2.3.2 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 6: *Parameters for challenge response*

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

2.4. CHALLENGE SUCCESS

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

2.4.1 Request

The command has no parameters.

2.4.2 Response

There is no response.

2.5. CLOUD SESSION

This command is used to request a cloud session [TBD]

2.5.1 Command

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*token length*	The number of bytes in the token; may be 0
2	varies	uint8_t	*token*	The session token
	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 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 version >= 5

Table 7: *Parameters for Cloud Session request*

2.5.2 Response result

The parameters for the connection response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*success*	0 if failed, otherwise successful
1	1	uint8_t	*status*	See *Table 9: Cloud status enumeration*
2	1	uint16_t	*token length*	The number of bytes in the client token GUID; may be 0
	varies	uint8_t[]	*token*	The client token GUID

Table 8: *Parameters for Cloud Session Response*

The cloud status types are:

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

Table 9: *Cloud status enumeration*

2.6. CONNECT

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

2.6.1 Request

The parameters of the request body are:

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

Table 10: *Parameters for Connection request*

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

2.6.2 Response

The parameters for the connection response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*connectionType*	See *Table 12: Connection types enumeration*
1	32	uint8_t[32]	*publicKey*	The public key to use for the connection

Table 11: *Parameters for Connection Response*

The connection types are:

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

Table 12: *Connection types enumeration*

The application sends the response, with its publicKey (see section *1.2 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:

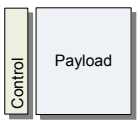

Figure 2: *The format of a frame*

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:

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

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.

2.7. 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.

2.7.1 Request

The command has no parameters.

2.7.2 Response

There is no response.

2.8. FILE DOWNLOAD

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

2.8.1 Request

The parameters of the request body are:

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	uint12_t	*length*	The number of bytes to follow (can be 0)
	varies	uint8_t[length]	*bytes*	The bytes of this file chunk

Table 13: *Parameters for File Download request*

2.8.2 Response

There is no response [?TBD?]

2.9. LOG

This command is used to request the Vector TBD logging

2.9.1 Request

The parameters of the request body are:

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

Table 14: *Parameters for Log request*

Each filter entry has the following structure:

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

Table 15: *Log filter*

2.9.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*exit code*	
1	4	uint32_t	*file id*	

Table 16: *Parameters for Log Response*

2.10. NONCE

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

2.10.1 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 17: *Parameters for Nonce request*

2.10.2 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

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

2.11. OTA UPDATE

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

2.11.1 Request

The parameters of the request body are:

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

Table 18: *Parameters for OTA request*

2.11.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*status*	See *Table 20: 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

Table 19: *Parameters for OTA Response*

The OTA status are:

Index	Meaning
0	idle
1	unknown
2	in progress
3	complete
4	rebooting
5	error

Table 20: *OTA status enumeration*

2.12. RESPONSE

It is not known why this message will be sent.

Table 21: *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*	
	varies	uint8_t [length]	*bytes*	

2.13. SDK PROXY

This command is used to pass the gRPC/protobufs messages to Vector over Bluetooth LE. It effectively wraps a HTTP request/response.

2.13.1 Request

The parameters of the request body are:

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)

Table 22: *Parameters for the SDK proxy request*

2.13.2 Response

The parameters for the response message:

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)

Table 23: *Parameters for the SDK proxy Response*

2.14. SSH

This command is used to request the Vector allow SSH. It is not known which version of the Vector support SSH, or whether it is enabled in the release.

2.14.1 Request

The parameters for the request message:

Table 24: *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 25: *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

2.14.2 Response

The response has no parameters

2.15. STATUS

This command is used to request the Vector act basic info.

2.15.1 Request

The request has no parameters

2.15.2 Response

The parameters for the response message:

***Table 26:** 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 27: 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*	
	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 27:** WiFi state enumeration*

Index	Meaning
0	Unknown
1	Online
2	Connected
3	Disconnected

2.16. WIFI ACCESS POINT

This command is used to request that the Vector act as a WiFi access point.

2.16.1 Request

The parameters of the request body are:

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

Table 28: *Parameters for WiFi Access Point request*

2.16.2 Response

The parameters for the response message:

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

Table 29: *Parameters for WiFi Access Point Response*

2.17. WIFI CONNECT

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

2.17.1 Request

The parameters for the request message:

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 31: WiFi authentication types enumeration*
	1	uint8_t	*hidden*	0 the access point is not hidden; 1 it is hidden

***Table 30:** Parameters for WiFi Connect request*

The WiFi authentication types are:

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

***Table 31:** WiFi authentication types enumeration*

2.17.2 Response

The parameters for the response message:

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 27: WiFi state enumeration*
	1	uint8_t	*connect result*	version >= 3

***Table 32:** Parameters for WiFi Connect command*

2.18. WIFI FORGET

This command is used to request the Vectors forget a WiFi SSID.

2.18.1 Request

The parameters for the request message:

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

Table 33: *Parameters for WiFi Forget request*

2.18.2 Response

The parameters for the response message:

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

Table 34: *Parameters for WiFi Forget response*

2.19. WIFI IP ADDRESS

This command is used to request the Vectors WiFi IP address.

2.19.1 Request

The request has no parameters

2.19.2 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 35: *Parameters for WiFi IP Address response*

2.20. WIFI SCAN

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

2.20.1 Request

The command has no parameters.

2.20.2 Response

The parameters for the response message:

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

Table 36: *Parameters for WiFi scan response*

Each access point has the following structure:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*auth type*	The type of authentication to employ; see *Table 31: 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

Table 37: *Parameters access point structure*

Appendices

- ABBREVIATIONS, ACRONYMS, & GLOSSARY. This appendix provides a gloss of terms, abbreviations, and acronyms.
- TOOL CHAIN. This appendix lists the tools known or suspected to have been used by Anki to create, and customize the Vector, and for the servers. Tools that can be used to analyze vector
- 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 38: 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	data analytics service ?
DFU	device firmware upgrade
EEPROM	electrical-erasable programmable read-only memory
ESD	electro-static discharge
ESN	electronic serial number
GPIO	general purpose IO
I2C	Inter-IC communication
IMU	inertial measurement unit
IR	infrared
JTAG	Joint Test Action Group
LCD	liquid crystal display
LED	light emitting diode
LUKS	linux unified key setup
MEMS	micro-electromechanical systems
MISO	Master-in, slave-out
MOSI	Master-out, slave-in
OLED	organic light-emitting diode display
OTA	over the air updates
RRT	rapidly-expanding random tree
SCLK	(I2C) serial clock

SDA	(I2C) serial data
SDK	software development kit
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
TTS	text to speech
UART	universal asynchronous receiver/transmitter
USB	universal serial bus
UUID	universally unique identifier
vic	short Victor (Vector's working name)

Table 39: *Glossary of common terms and phrases*

Phrase	Description
A*	A path finding algorithm
attitude	orientation
bootloader	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.
capacitive touch	
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.
control	motors and forces to move where and how it is told to. (smooth arcs)
D*-lite	A path-finding algorithm
flash	A type of persistent (non-volatile) storage media.
guidance	desired path
navigation	knowing where it is in the map
nonce	An initially random number, incremented after each use .
pose	position and orientation of an object relative to a coordinate system
power source	Where the electric energy comes from.
path planning	smooth arcs and line segments
rapidly-expanding random tree	A path-finding algorithm
service (Bluetooth LE)	A key-value table, grouped together for a common purrpose. A service is uniquely identified by its UUID.
universally unique identifier (UUID)	A 128bits number that is unique.

APPENDIX B

Tool chain

This appendix tries to capture the tools that Anki is known or suspected to have used for the Anki Vector and its cloud server.

Note: Several of these the licenses requiring Anki to post their versions of the GPL tools, and their modification, Anki never did. Qualcomm may have; as the license requirement only to those their customer, they may have provided the changes to them.

***Table 40:** Tools used by Anki*

Tool	Description
Acapela	Vector uses Acapela's text to speech synthesizer, and the Ben voice. https://www.acapela-group.com/
Advanced Linux Sound Architecture (alsa)	The audio system https://www.alsa-project.org
Amazon Alexa	A set of software tools that allows Vector to integrate Alexa voice commands, probably in the AMAZONLITE distribution https://developer.amazon.com/alexa-voice-service/sdk
Amazon Web services	used on the server https://aws.amazon.com/
android boot-loader	Vector uses the Android Boot-loader;
AudioKinetic Wwise[1]	Used to craft the sounds https://www.audiokinetic.com/products/wwise/
clang	A C/C++ compiler, part of the LLVM family https://clang.llvm.org
bluez5	Bluetooth LE support http://www.bluez.org/
busybox	The shell on the Anki Vector linux https://busybox.net
chromium update	?
civetweb	The embedded webserver that allows Mobile apps and the python SDK to communicate with Vector. https://github.com/civetweb/civetweb
connman	Connection manager for WiFi https://01.org/connman
GNU C Compiler (gcc)	GCC version 4.9.3 was used to compile the kernel
golang	on the server
Google RPC (gRPC)	A "remote procedure call" standard, that allows mobile apps and the python SDK

[1] https://blog.audiokinetic.com/interactive-audio-brings-cozmo-to-life/?hsFormKey=227ccf4a650a1cffd6562c16d655a0ef

	to communicate with Vector. https://grpc.io/docs/quickstart/cpp/
hdr-histogram	Unknown use https://github.com/HdrHistogram/HdrHistogram
libchromatix	Qualcomm camera support
libsodium	Cryptography library suitable for the small packet size in Bluetooth LE connections. Used to encrypt the mobile applications Bluetooth LE connection with Vector. https://github.com/jedisct1/libsodium
linux, yocto[2]	The family of linux distribution used for the Anki Vector (v3.18.66)
linux	on the server
linux unified key storage (LUKS)	
Maya	A character animation tool set, used to design the look and movements of Cozmo and Vector. The tool emitted the animation scripts.
mpg123	A MPEG audio decoder and player. This is needed by Alexa; other uses are unknown. https://www.mpg123.de/index.shtml
ogg vorbis	Audio codec https://xiph.org/vorbis
open CV	Used for the first-level image processing – to locate faces, hands, and possibly accessory symbols. https://opencv.org/
openssl	used to validate firmware update signature https://www.openssl.org
opkg	Package manager, from yocto https://git.yoctoproject.org/cgit/cgit.cgi/opkg/
Opus codec	Audio codec; may be used to encode speech sent to servers http://opus-codec.org/
perl	A programming language, on Victor https://www.perl.org
protobuf	Used to describe the format/encoding of data sent over gRPC to and from Vector. This is used by mobile and python SDK, as well as on the server. https://developers.google.com/protocol-buffers
Pryon	The recognition for the Alexa keyword at least the file system includes the same model as distributed in AMAZONLITE https://www.pryon.com/company/
python	A programming language and framework, used with desktop tools to communicate with Vector. Vector has python installed. Probably used on the server as well. https://www.python.org
Qualcomm TBD	Qualcomm's device drivers, and other kit appears to be used.
Segger ICD	A high-end ARM compatible in-circuit debugging probe. Rumoured to have been used by Anki engineers, probably with the STM32F030 https://www.segger.com/products/debug-probes/j-link/

[2] https://www.designnews.com/electronics-test/lessons-after-failure-anki-robotics/140103493460822

Sensory	Includes recognition for hey vector and Alexa wake word by Sensory, Inc. https://en.wikipedia.org/wiki/Sensory,_Inc.
SQLite	This is needed by Alexa; other uses are unknown https://www.sqlite.org/index.html
systemd	Used by Vector to launch the internal services https://www.freedesktop.org/software/systemd/
tensor flow lite (TFLite)	Probably used to recognize the object marker symbols, and maybe hands https://www.tensorflow.org/lite/microcontrollers/get_started

Other tools, useful for analyzing and patching Vector:

Table 41: *Tools that can be used to analyze and patch Vector*

Toool	**Description**
Segger ICD	An education version of the J-Link, suitable for the STM32F030, can be found on ebay for <$60 https://www.segger.com/products/debug-probes/j-link/
ST-Link (v3)	Suitable for extracting the STM32F030 and installing patched firmware; $35 https://www.st.com/en/development-tools/stlink-v3set.html
TI BLE sniffer	$50 http://www.ti.com/tool/CC2540EMK-USB https://www.ti.com/tool/PACKET-SNIFFER
Wireshark	To decode what is said to the servers https://support.citrix.com/article/CTX116557

APPENDIX C

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.

3. CUBE SERVICES

times and other feature parameters:

Service	UUID[3]	Description & Notes
Device Info Service[4]	$180A_{16}$	Provides device and unit specific info – it's manufacturer, model number, hardware and firmware versions
Generic Access Profile[5]	1800_{16}	The device name, and preferred connection parameters
Generic Attribute Transport[6]	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

Table 42: *The Bluetooth LE services*

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)

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

Table 43: *The Cube's Device info settings*

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

3.1. CUBE'S ACCELEROMETER SERVICE

Values are little-endian, except where otherwise stated.

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*
		uint16_t	
9590BA9C-5140-92B5-1844-5F9D681557A4$_{16}$	Write		Unknown

***Table 44:** Cube's accelerometer service characteristics*

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

4. VECTOR SERVICES SERVICE

times and other feature parameters:

Service	UUID[7]	Description & Notes
Generic Access Profile	1800_{16}	The device name, and preferred connection parameters
Generic Attribute Transport	1801_{16}	Provides access to the services.
Vectors Serial Service	$FEE3_{16}$	The service with which we can talk to Vector.

***Table 45:** Vector's Bluetooth LE services*

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

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

***Table 46:** The Vector's Device info settings*

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

4.1. VECTORS SERIAL SERVICE

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

Table 47: *Vector's serial service characteristics*