TECHNICAL DEEP DIVE

Anki Vector

A LOVE LETTER TO THE LITTLE DUDE

AUTHOR RANDALL MAAS

OVERVIEW This is fascicle explores how the Anki Vector was realized in hardware and software.

sketch by Steph Dere

RANDALL MAAS has spent decades in Washington and Minnesota. He consults in embedded systems development, especially medical devices. Before that he did a lot of other things… like everyone else in the software industry. He is also interested in geophysical models, formal semantics, model theory and compilers.

You can contact him at randym@randym.name.

LinkedIn: http://www.linkedin.com/pub/randall-maas/9/838/8b1

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

Preface

The Anki Vector is a charming little robot – cute, playful, with a slightly mischievous character. It is everything I ever wanted to create in a bot. Sadly Anki went defunct shortly after releasing Vector.

This book is my attempt to understand the Anki Vector and its construction. The book is based on speculation. Speculation informed by Anki's SDKs, blog posts, patents and FCC filings; by articles about Anki, presentations by Anki employees; by PCB photos, and foo'd hardware from others; and by experience with the parts, and areas.

1.1. CUSTOMIZATION AND PATCHING

What can be customized – or patched – in Vector?

- The firmware in the main processor, that will be dicussed in the rest of the document
- The base-board is not field updatable, without expertise. The STM32F030 can be extracted with a ST-Link ($25), and the the firmware can be disassembled — the microcontroller is conducive to this. Shy of recreating the firmware, the patches replace a key instruction here and there with a jump to the patch, created in assembly (most likely) code to fix or add feature, then jump back. (STM32's come with a boot loader, but the ST-Link is easier and inexpensive.)
- The cube firmware can be updated, but that seems both hard and not useful.

2. ORGANIZATION OF THIS DOCUMENT

- CHAPTER 1: PREFACE. This chapter describes the other chapters.
- CHAPTER 2: OVERVIEW OF VECTOR'S ARCHITECTURE. Introduces the overall design of the Anki Vector.

PART I: ELECTRICAL DESIGN.

- CHAPTER 3: VECTOR'S ELECTRICAL DESIGN. A detailed look at the electrical design of Vector.
- CHAPTER 4: ACCESSORY ELECTRICAL DESIGN. A look at the electrical design of Vectors accessories.

PART II: BASIC OPERATION.

- CHAPTER 6: COMMUNICATION. A look at the communication stack in Vector.
- CHAPTER 7: BLUETOOTH LE. The Bluetooth LE protocol that Vector responds to.
- CHAPTER 8: CLOUD. A look at the electrical design of Vectors accessories.

REFERENCES AND RESOURCES. This provides further reading and referenced documents.

APPENDICES: The appendices provides extra material

- APPENDIX A: ABBREVIATIONS, ACRONYMS, & GLOSSARY. This appendix provides a gloss of terms, abbreviations, and acronyms.

- APPENDIX B: 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
- APPENDIX C: BLUETOOTH LE PROTOCOLS. This appendix provides information on the Bluetooth LE interfaces to the companion Cube, and to Anki Vector
- APPENDIX D: SERVERS. This appendix provides the servers that the Anki Vector and App contacts
- APPENDIX E: PHRASES. This appendix reproduces the phrases that the Vector keys off of.
- APPENDIX F: FEATURES. This appendix enumerates the Vector OS "features" that can be enabled and disabled.
- APPENDIX G: FILE SYSTEM. This appendix lists the key files that are baked into the system.
- APPENDIX H: PLEO. This appendix gives a brief overview of the Pleo animatronic dinosaur.

Note: I use many diagrams from Cozmo literature. They're close enough

CHAPTER 2

Overview of Vector

Anki Vector is a cute, palm-sized robot; a buddy with a playful, slightly mischievous character. This chapter is provides an overview of Vector:

- Overview
- Ancestry: Cozmo

3. OVERVIEW

Vector is an emotionally expressive animatronic robot that we all love.

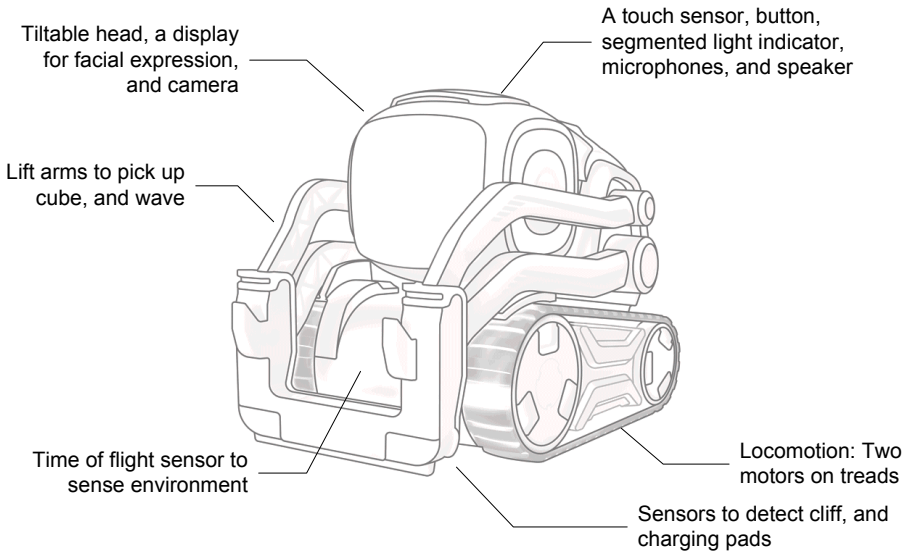


Figure 1: *Vector's main features*

He can express emotions thru expressive eyes (on an LCD display), raising and lower his head, sounds, wiggling his body (by using his treads), or lifting his arms… or shaking them.

Vector can sense surrounding environment, interact and respond to it. Recognize his name[1], follow the gaze of a person looking at him, and seek petting.[2]

3.1. FEATURES

Although cute, small, and affordable,[3] Vector's design is structured like many other robots.

[1] Vector can't be individually named.
[2] Admittedly this is a bit hit and miss.
[3] Although priced as an expensive toy, this feature set in a robot is usually an order of magnitude more, usually with less quality.

He has a set of operator inputs:

- A touch sensor is used detect petting
- Internal microphone(s) to listen, hear commands and ambient activity level
- A button that is used to turn Vector on, to cause him to listen – or to be quiet (and not listen), to reset him (wiping out his personality and robot-specific information).
- He can detect his arms being raised and lowered.[4]

He has a set of indicators/annuciators:

- Segmented lights on Vectors backpack are used to indicate when he is on, needs the charger, has heard the wake word, is talking to the Cloud, can't detect WiFi, is booting, is resetting (wiping out his personality and robot-specific information).
- An LCD display, primarily to show eyes of a face. Robot eyes were Anki's strongest piece of imagery. Vector smiles and shows a range of expressions with his eyes.
- Speaker for cute sounds and speech synthesis

He has other means to express affect as well:

- His head can be tilted up and down to represent sadness, happiness, etc.
- His arms flail to represent frustration
- He can uses his treads to shake or wiggle, usually to express happiness or embarrassment

He has environmental sensors:

- A camera is used to map the area, detect and identify objects and faces.
- Fist-bump and being lifted can be detected using an internal IMU
- A forward facing "time of flight" proximity sensor aids in mapping and object avoidance
- Ground sensing proximity sensors that are used to detect cliffs and the edge of his area; these may also be used in following lines on his charger.

Internal sensing includes:

- Battery voltage, charging; charging temperature
- IMU for orientation position (6-axis)
- Encoders provide feedback on motor rotation

Other articulation & actuators:

- Vector drives using two independent treads to do skid-steering
- Using his arms Vector can lift, or flip a cube; he can pop a wheelie, or lift himself over a small obstacle.
- Vector can raise and lower his head

Communication (other than user facing):

- Communication with the external world is thru WiFi and Bluetooth LE.
- Internally RS-232 (CMOS levels) and USB

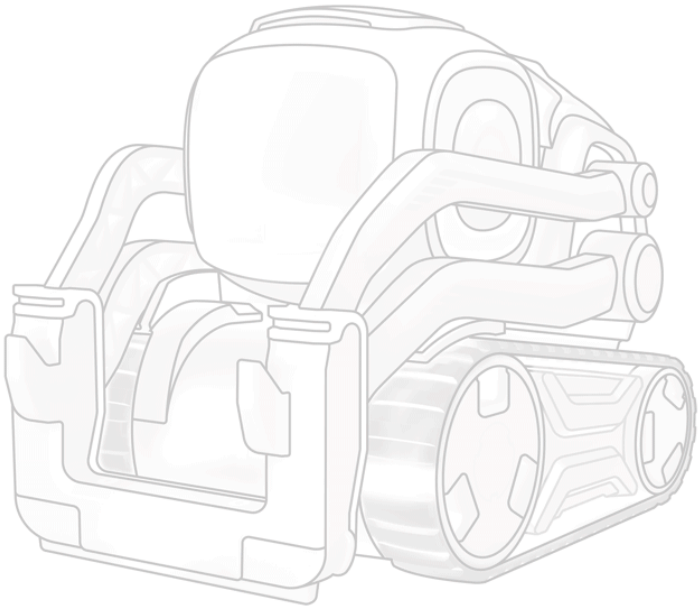
[4] and possibly his head?

PART I

Electronics Design

This part provides an overview of the design of the electronics in Vector and his accessories

- VECTOR'S ELECTRICAL DESIGN. A detailed look at the electrical design of Vector.
- ACCESSORY ELECTRICAL DESIGN. A look at the electrical design of Vectors accessories.



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

CHAPTER 3

Electronics design description

This chapter describes the electronic design of the Anki Vector:

- Design Overview
- Detailed design of the main board
- Detailed design of the base-board
- Power characteristics

4. DESIGN OVERVIEW

Vector's design includes numerous some to sense and interact with his environment, other to interact with people and express emotion and behaviour.

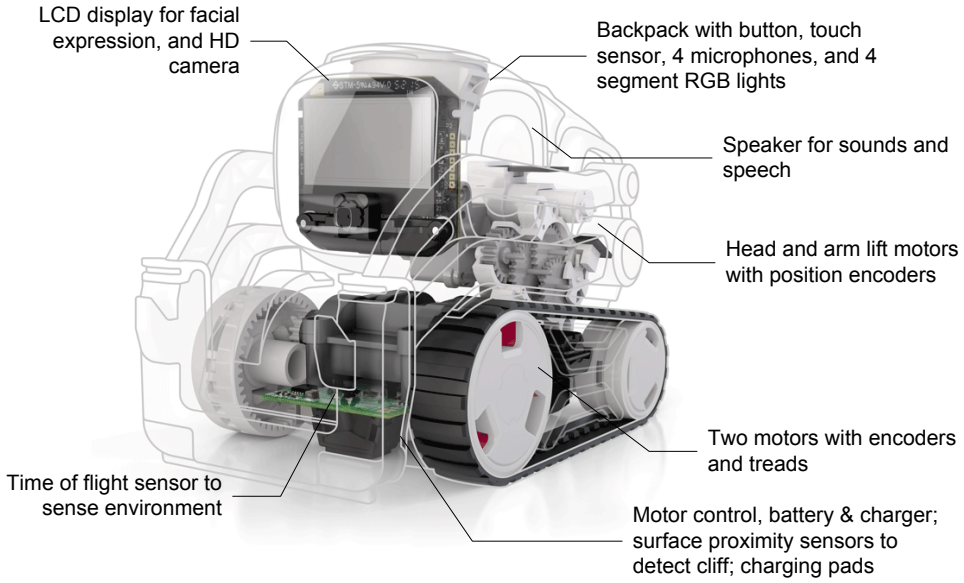


Figure 2: *Vector's main elements*

Vector's functional elements are:

Element	Description
backpack	The top of Vector, where he has a button, segmented lights, and a touch sensor.
battery	There is an internal battery pack (3.7v 320 mAh) that is used as Vector's source of energy.
button	A momentary push button is used to turn Vector on, to cause him to listen – or to be quiet (and not listen) – to reset him (wiping out his personality and robot-specific information).
camera	Vector uses an HD camera to visualize his environment, and recognize his human companions.
charging pad	Two pads on the bottom are used to replenish the energy in the battery pack from the dock.
LCD display	An IPS LCD, with an active area is 23.2mm x 12.1mm. It has a resolution of 184 x 96 pixels, with RGB565 color.
microphones	There are 4 internal MEMS microphone(s) to listen to commands and ambient activity level. Employs beam forming to localize sounds.
motors & encoders	There are four motors with single-step optical encoders to measure their position and approximate speed. One motor controls the tilt of the head assembly. Another controls the lift of his arms. Two are used to drive him in a skid-steering fashion.
segmented RGB lights	There are 4 LEDs used to indicate when he is on, needs the charger, has heard the wake word, is talking to the Cloud, can't detect WiFi, is booting, is resetting (wiping out his personality and robot-specific information).
speaker	A speaker is used to play sounds, and for speech synthesis
surface proximity sensors	4 infrared proximity sensors are used to detect the surface beneath Vector – and to detect drop offs ("cliffs") at the edge of his driving area.
time of flight sensor	A time of flight sensor is used to aid in mapping (by measuring distances) and object avoidance.
touch sensor	A touch allows Vector to detect petting and other attention.

Table 1: *Vectors main elements*

Vector has 6 circuit boards. The main two boards are the head-board where the major of Vectors processing occurs, and the base-bard, which drives the motors and connects to the other boards.

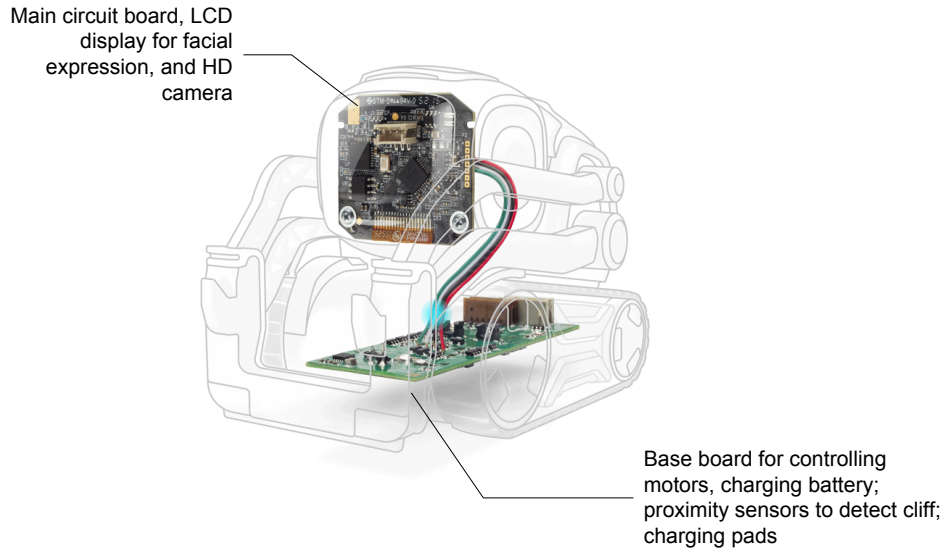


Figure 3: *Vector's main microcontroller circuit boards*

The table below summarizes the boards:

Circuit Board	Description
backpack board	The backpack board has 4 RGB LEDs, 4 MEMS microphones, a touch wire, and a button. This board connects to the base-board.
base-board	Drives the motor. power management battery charger
encoder-boards	The two encoder boards have single opto-coupler encoder each. The encoder is used to monitor the position of the arms and head, either as driven by the motor, or by a person manipulating them.
head-board	The head board includes the main processor, flash & RAM memory storage, an IMU, and a PMIC. The WiFi and Bluetooth LE are built into the processor. The camera and LCD are attached to the board, thru a flex tape. The speaker is also attached to this board.
time of flight sensor board	The time of flight sensor is on a separate board, allowing it to be mounted in Vectors front.

Table 2: *Vector's circuit boards*

4.1. POWER SOURCE AND DISTRIBUTION TREE

Vector is powered by a rechargeable battery pack, and the energy is distributed by the base-board:

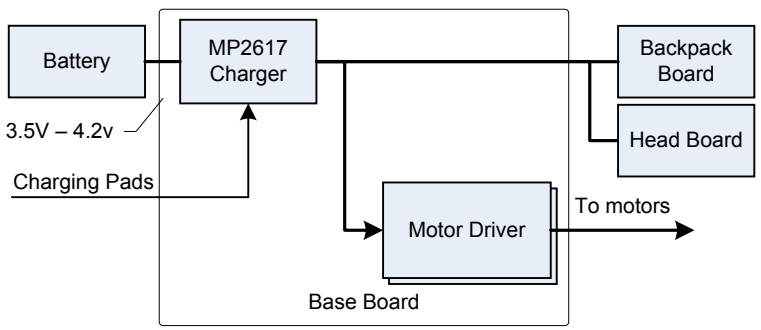


Figure 4: *Power distribution*

The MP2617 is a central element to managing the battery. It acts as a battery charger, a power switch and power converter for the whole system.

- When Vector is going into an *off* state – such as running too low on power, going into a ship state before first use, or has been turned off by a human companion – the MP2617 charger and power converted can be signaled to turn off
- When Vector is turned off the boards are not energized. The exception is that the high side of the push button is connected to the battery. When closed, the signals the MP2617 to connect the battery to the rest of the system, powering it up.
- The MP2617 is also responsible for charging the battery. There are two pads that mate the dock to supply energy to charge the battery.

In many rechargeable lithium ion battery systems there is a coulomb counter to track the state of charge. Vector does not have one. The need for recharge is triggered solely on the battery voltage.

5. THE BACKPACK BOARD

The backpack board is effectively daughter board to the base-board. It provides extra IO and a couple of smart peripherals:

- 4 RGB LEDs, with a 74AHC164 to drive them. These are designed D1, D2, D3, D4
- 4 MEMS microphones, with SPI interfaces. These are designated MK1, MK2, MK3, MK4.
- A touch-sensing wire,
- A momentary-contact push button,

5.1. BACKPACK CONNECTION

The backpack connection includes

- Power and ground connections. This includes connection to the battery rail.
- The touch wire as an analog signal to the base-board
- A quasi digital signal out from the momentary push buton
- (at least) Two chip selects
- A SPI-like set of clock, master-out-slave-in (MOSI) and *two* master-in-slave-out (MISO) signals

5.2. OPERATION

The touch sensor conditioning and sensing is handled by the base-board.

The push-button is wired to the battery. When pressed, the other side of the push button signals both base-board microcontroller, and (if Vector is off) the charger chip to connect power.

The 74AHC164 serial-shift-register is basically a GPIO expander. It takes a chip select, clock signal and serial digital input. The inputs determine the state of 8 digital outputs used to control the RGB LEDs. More on this below.

Each of the 4 MEMS microphones take a chip select, clock signal, and provide a serial digital output. The clock signal (and one of the chip selects) is shared with the 74AHC164.

The base-board sets the digital outputs, and reads 2 microphones at a time. It reads all four microphones by alternating the chip selects to select which two are being accessed. (This will be discussed in the base-board section).

5.2.1 The LED control

8 outputs are not enough to drive 4 RGB LEDs (each with 3 inputs) independently. 3 of the LEDs are always the same colour – but illuminated independently. The 4th LED has a different colours, and is illuminated independently.

- D1 has separate red and green signals from the 74AHC164. It may share blue with the others.
- 3 signals from the 74AHC164 – Red, Green, and Blue – are shared for D2, D3, D4.[5]
- D2, D3, and D4 each have individual bottom drives

With care, the LEDs can be individually turned on and off (the low sides), and selected for a colour (the red, green, and blue signals).

6. THE BASE-BOARD

The base board is mostly a smart IO expander. It is based on an STM32F030 which talks to the head-board via RS-232 (estimated to be 2Mbps).

Uses an SPI MCLK to clock out the state, and MOSI to send the state of that IO channel

7. THE HEAD-BOARD (THE MAIN PROCESSOR BOARD)

The head-board is based on a Qualcomm APQ8009 (Snapdragon 212). It has a Kingston 04EMCP04-NL3DM627 mixed memory chip with 4 GB FLASH and 512MB RAM. This processor is a quad-core Arm A7 (32-bit) CPU, and includes 802.11ac Wi-Fi and Bluetooth LE transceivers in the package.

Most processing is done on the head-board.

8. RESOURCES

Diodes, Inc, *74AGC164 8-Bit Parallel-Out Serial Shift Registers*, Rev 2, 2015 Aug
https://www.diodes.com/assets/Datasheets/74AHC164.pdf

Monolithic Power, *MP2617A, MP2617B 3A Switching Charger with NVDC Power Path Management for Single Cell Li+ Battery,* Rev 1.22 2017 Jun 29
https://www.monolithicpower.com/pub/media/document/MP2617A_MP2617B_r1.22.pdf

ST Microelectronics
https://www.st.com/content/ccc/resource/technical/document/application_note/group0/ed/0d/4d/87/04/1d/45/e5/DM00445657/files/DM00445657.pdf/jcr:content/translations/en.DM00445657.pdf

https://www.st.com/en/embedded-software/32f0-touch-lib.html

https://hsel.co.uk/2016/05/22/stm32f0-software-capacitive-touch/

https://github.com/pyrohaz/STM32F0-SoftTouch

[5] If I'm seeing the chip right, the ground, green, and blue are wired together but that doesn't make sense in the truth-table to get the effect of the LED patterns

CHAPTER 4

Accessory Electronics design description

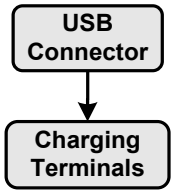
This chapter describes the electronic design of the Anki Vector accessories:

- The charging station
- The companion cube

9. CHARGING STATION

The charging station is intended to provide energy to the Vector, allowing it to recharge. The charging station has a USB cable that plugs into an outlet adapter or battery. The adapter or battery supplies power to the charging station. The base of the station has two terminals to supp +5V (from the power adapter) to Vector, allowing him to recharge.

Figure 5: *Charging station block diagram*



The charging station has an optical marker used by Vector to identify the charging station and its pose (see chapter TBD).

10. CUBE

This section describes the companion cube accessory. The companion cube is a small toy for Vector play with. He can fetch it, roll it, and use it to pop-wheelies. Each face of the cube has a unique optical marker used by Vector to identify the cube and its pose (see chapter TBD).

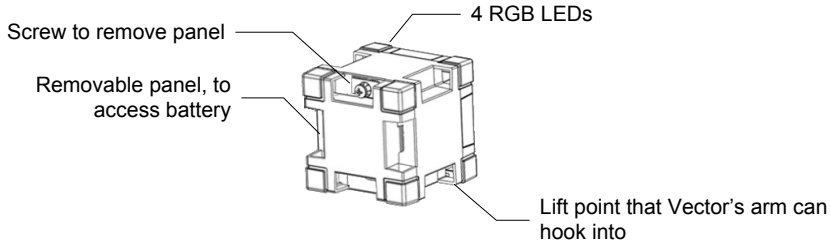


Figure 6: *Cube's main features*

Although the companion cube is powered, this is not used for localization or pose. The electronics are only used to flash lights for his owner, and to detect when a person taps, moves the cube or changes the orientation.

The cube has holes near the corners to allow the lift to engage, allowing Vector to lift the cube. Not all corners have such holes. The top – the side with the multicolour LEDs – does not have these. Vector is able to recognize the cubes orientation by symbols on each face, and to flip the cube so that it can lift it.

The electronics in the cube are conventional for a small Bluetooth LE accessory:

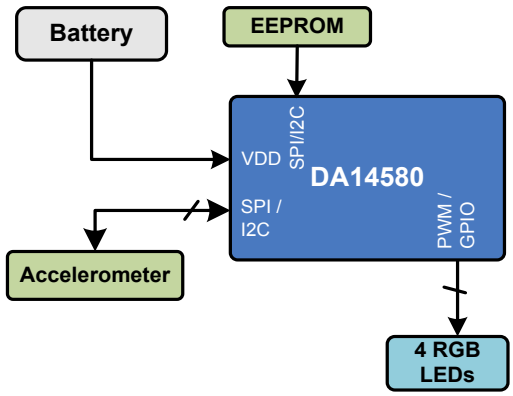


Figure 7: *Block diagram of the Cube's electronics*

The Cube's electronic design includes the following elements:

Element	Description
accelerometer	Used to detect movement and taps of the cube.
battery	The cube is powered by a 1.5 volt N / E90 / LR1 battery cell.[6]
crystal	The crystal provides the accurate frequency reference used by the Bluetooth LE radio.
Dialog DA14580	This is the Bluetooth LE module (transmitter/receiver, as well as microcontroller and protocol implementation).
EEPROM	The EEPROM holds the updatable application firmware.
RGB LEDs	There are 4 RGB LEDs. They can flash and blink. Unlike the backpack LEDs, two LEDs can have independent colors.

Table 3: *The Cube's electronic design elements*

The communication protocol is given appendix C.

10.1. OVER THE AIR FIELD UPDATES

The DA14580 has a minimal ROM bootloader that initializes hardware, moves a secondary bootloader from "One Time Programmable" ROM (OTP) into SRAM, before passing control to it. The firmware is executed from SRAM to reduce power consumption. The secondary bootloader loads the application firmware from I2C or SPI EEPROM or flash to SRAM and pass control to it.

[6] The size is similar to the A23 battery, which will damage the cubes electronics.

If the application passes control back to the bootloader – or there isn't a valid application in EEPROM –a new application can be downloaded. The bootloader uses a different set of services and characteristics to support the bootloading process.

10.2. RESOURCES

Dialog, *SmartBond™ DA14580 and DA14583*
https://www.dialog-semiconductor.com/products/connectivity/bluetooth-low-energy/smartbond-da14580-and-da14583

Dialog, *DA14580 Low Power Bluetooth Smart SoC, v3.1, 2015 Jan 29*

Dialog, *UM-B-012 User manual DA14580/581/583 Creation of a secondary bootloader*, CFR0012-00 Rev 2, 2016 Aug 24
Dialog, *Application note: DA1458x using SUOTA,* AN-B-10, Rev 1, 2016-Dec-2
https://www.dialog-semiconductor.com/sites/default/files/an-b-010_da14580_using_suota_0.pdf

PART II

Basic Operation

This part provides an overview of the design of the electronics in Vector and his accessories

- COMMUNICATION. A look at the communication stack in Vector.
- BLUETOOTH LE. The Bluetooth LE protocol that Vector responds to.
- CLOUD. A look at how Vector syncs with the cloud

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

CHAPTER 5

Communication

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

11. 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 rough stack:

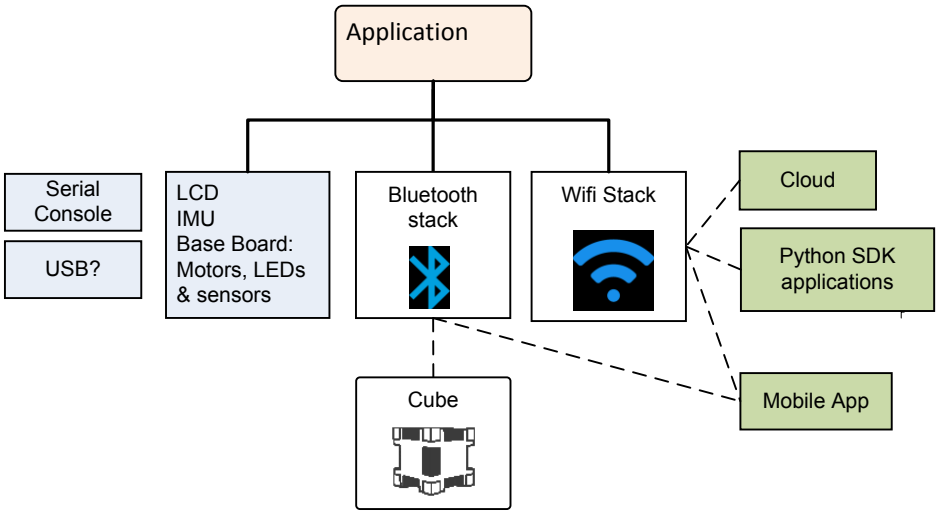


Figure 8: *The overall communication infrastructure*

12. INTERNAL COMMUNICATION WITH PERIPHERALS

12.1. COMMUNICATION WITH THE BASE-BOARD

The head board communicates with the base board using a serial interface[7]

Data rate: estimated to be ~2Mbps

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

The messages from the Head to the base 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.

12.2. SERIAL BOOT CONSOLE

The head-board includes a 115200, 8 data bits no parity, 1 stop bit. Only prints the boot console.

12.3. 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.

The /etc/initscripts/usb file enables the USB and the usual functionfs adb. This launches ADB (DIAG + MODEM + QMI_RMNET + ADB)

Melanie_t reports this not working, not enabled.

13. BLUETOOTH LE

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.

Bluetooth LE is used to communicate with the companion Cube accessory: to detect its movement, taps, and to light it up.

---

[7] /dev/tty unknown

CHAPTER 7

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 C.

14. 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:

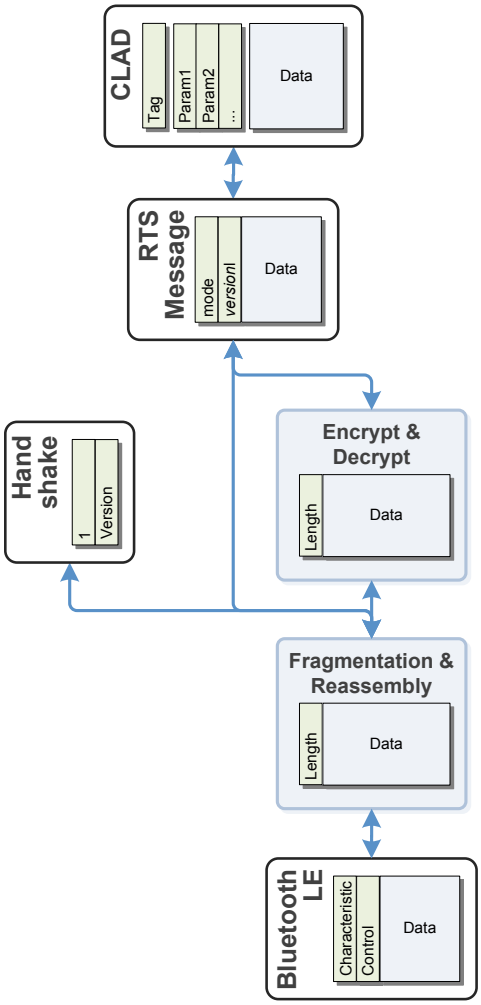


Figure 9: *Overview of encryption and fragmentation stack*

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,

14.1. 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 4: *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 *14.3.1 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 *14.3.1 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.

14.2. 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:

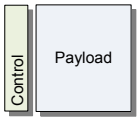


Figure 10: *The format of a frame*

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.

14.3. 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 5: *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

14.3.1 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);
```

14.3.2 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?}

14.3.3 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.

14.4. 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:

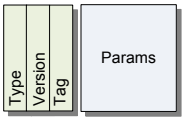


Figure 11: *The format of an RTS frame*

- 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.

15. 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 6: *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

15.1. APPLICATION CONNECTION ID

?

15.1.1 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 7: *Parameters for Application Connection Id request*

15.1.2 Response

There is no response.

15.2. CANCEL PAIRING

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

15.2.1 Request

The command has no parameters.

15.2.2 Response

There is no response.

15.3. CHALLENGE

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

15.3.1 Request

The parameters of the request body are:

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

Table 8: *Parameters for challenge request*

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

15.3.2 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 9: *Parameters for challenge response*

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

15.4. CHALLENGE SUCCESS

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

15.4.1 Request

The command has no parameters.

15.4.2 Response

There is no response.

15.5. CLOUD SESSION

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

15.5.1 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 10: *Parameters for Cloud Session request*

15.5.2 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 12: 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 11: *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 12: *Cloud status enumeration*

15.6. CONNECT

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

15.6.1 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 13:** Parameters for Connection request*

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

15.6.2 Response

The parameters for the connection response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*connectionType*	See *Table 15: Connection types enumeration*
1	32	uint8_t[32]	*publicKey*	The public key to use for the connection

***Table 14:** Parameters for Connection Response*

The connection types are:

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

***Table 15:** Connection types enumeration*

The application sends the response, with its publicKey (see section *14.2 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:

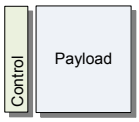


***Figure 10:** The format of a frame*

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.

15.7. 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.

15.7.1 Request

The command has no parameters.

15.7.2 Response

There is no response.

15.8. FILE DOWNLOAD

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

15.8.1 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 16: *Parameters for File Download request*

15.8.2 Response

There is no response [?TBD?]

15.9. LOG

This command is used to request the Vector TBD logging

15.9.1 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 17: *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 18: *Log filter*

15.9.2 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 19: *Parameters for Log Response*

15.10. NONCE

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

15.10.1 Request

The parameters for the nonce request message:

Table 20: Parameters for Nonce request

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

15.10.2 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.

15.11. OTA UPDATE

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

15.11.1 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 21: *Parameters for OTA request*

15.11.2 Response

The parameters for the response message:

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

Table 22: *Parameters for OTA Response*

The OTA status are:

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

Table 23: *OTA status enumeration*

15.12. RESPONSE

It is not known why this message will be sent.

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*	
	varies	uint8_t [length]	*bytes*	

15.13. SDK PROXY

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

15.13.1 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 25: *Parameters for the SDK proxy request*

15.13.2 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 26: *Parameters for the SDK proxy Response*

15.14. 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.

15.14.1 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

15.14.2 Response

The response has no parameters

15.15. STATUS

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

15.15.1 Request

The request has no parameters

15.15.2 Response

The parameters for the response message:

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*	
	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

***Table 29:** Parameters for Status Response*

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:

Index	Meaning
0	Unknown
1	Online
2	Connected
3	Disconnected

***Table 30:** WiFi state enumeration*

15.16. WIFI ACCESS POINT

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

15.16.1 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 31: *Parameters for WiFi Access Point request*

15.16.2 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 32: *Parameters for WiFi Access Point Response*

15.17. WIFI CONNECT

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

15.17.1 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 34: WiFi authentication types enumeration*
	1	uint8_t	*hidden*	0 the access point is not hidden; 1 it is hidden

***Table 33:** 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 34:** WiFi authentication types enumeration*

15.17.2 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 30: WiFi state enumeration*
	1	uint8_t	*connect result*	version >= 3

***Table 35:** Parameters for WiFi Connect command*

15.18. WIFI FORGET

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

15.18.1 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

15.18.2 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

15.19. WIFI IP ADDRESS

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

15.19.1 Request

The request has no parameters

15.19.2 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*

15.20. WIFI SCAN

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

15.20.1 Request

The command has no parameters.

15.20.2 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 39: *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 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

Table 40: *Parameters access point structure*

CHAPTER 8

The Cloud Services

This chapter is about the cloud

- The user's account settings and entitlements
- The robot's settings (user preferences)
- The robot's lifetime stats
- The crash uploader
- The firmware update process
- How to extract official program files

16. THE JDOC SERVER

16.1. THE USER ACCOUNT SETTINGS

Sent to the jdoc server

Schema: TBD

16.2. ROBOT SETTINGS

Sent to the jdoc server

These are an ad hoc set of locale preferences. The following are sent to the jdoc server by the mobile application, and retrieved by the robot:

***Table 41:** The robot settings.*

Key	units	Description & Notes
button_wakeword	?	
clock_24_hour	boolean	
default_location		
dist_is_metric		
eye_color		The color used for the eyes
locale		American English, UK English, Australian English, German, French, Japanese, etc.
master_volume		
temp_is_fahrenheit		
time_zone		

16.3. USER ENTITLEMENTS

Sent to the jdoc server

Schema: TBD

Appear to be the private and public keys

16.4. ROBOT LIFETIME STATS

Sent to the jdoc server

The following are held in the server, updated by the robot (I don't know if the robot has a local copy), and retrievable by the application. A JSON hash of the following:

***Table 42:** The robot lifetime stats schema*

Key	units	Description & Notes
Alive.seconds	seconds	Vectors age, since he was given preferences (a factory reset restarts this)
Stim.CumlPosDelta		Cumulative stimulation of some kind
BStat.AnimationPlayed	count	The number of animations played
BStat.BehaviorActivated	count	
BStat.AttemptedFistBump	count	The number of fist bumps (attempted)
BStat.FistBumpSuccess	count	
BStat.PettingBlissIncrease		
BStat.PettingReachedMaxBliss		
BStat.ReactedToCliff	count	
BStat.ReactedToEyeContact	count	
BStat.ReactedToMotion	count	
BStat.ReactedToSound	count	
BStat.ReactedToTriggerWord	count	
Feature.AI.DanceToTheBeat		
Feature.AI.Exploring		
Feature.AI.FistBump		
Feature.AI.GoHome		
Feature.AI.InTheAir		
Feature.AI.InteractWithFaces	count	The number of times recognized / interacted with faces
Feature.AI.Keepaway		
Feature.AI.ListeningForBeats		
Feature.AI.LowBattery		
Feature.AI.Observing		
Feature.AI.ObservingOnCharger		
Feature.AI.Onboarding		

Feature.AI.Sleeping		
Feature.AI.Petting		
Feature.AI.ReactToCliff		
Feature.AI.StuckOnEdge		
Feature.AI.UnmatchedVoiceIntent		
Feature.Voice.VC_Greeting		
FeatureType.Autonomous		
FeatureType.Failure		
FeatureType.Sleep		
FeatureType.Social		
FeatureType.Play		
FeatureType.Utility1		
Pet.ms	ms	The number of milliseconds petted?
Odom.LWheel		The left wheel odometer
'Odom.Rwheel		The right wheel odometer
Odom.Body		

17. DAS

DAS communication events, sent/received, home network stuff, etc.
seems to allow detailed reconstruction of the setup, configuration and interaction

Seems to gather mostly information from the mobile application – the name of each button pressed, screen displayed, error encoutered, etc.

Might get other activation information, used for the lifetime stats

References & Resources

Note: most references appear in the margins, significant references will appear at the end of their respective chapter.

18. CREDITS

Credit and thanks to Anki, C0RE, MelanieT for access to the partition file-system, and decode keys; board shots. Fictiv for board shots. The board shots that help identify parts on the board and inter-connection on the board.

19. REFERENCE DOCUMENTATION AND RESOURCES

19.1. ANKI

Anki, "*Vector Quick Start Guide,"* 293-00036 Rev: B, 2018

Anki, Molly Jameson & Daria Jerjomina, "*Cozmo: Animation pipeline for a physical robot,"* 2017 Game Developers conference

Anki, Nathaniel Monson, Andrew Stein, Daniel Casner *Reducing Burn-in of Displayed Images*, Patent US 20372659 A1, 2017 Dec 28

Anki, Daniel Casner, Lee Crippen, Hanns Tappeiner, Anthony Armenta, Kevin Yoon, *Map Related Acoustic Filtering by a Mobile Robot,* Patent US 0212441 A1, 2019 Jul 11

Anki, Andrew Stein, *Decoding Machine-Readable Optical codes with Aesthetic Component*, Patent US 9,607,199 B2, 2017 Mar. 28

19.2. OTHER

FCC *ID 2AAIC00010 internal photos*
https://fccid.io/2AAIC00010

FCC *ID 2AAIC00011 internal photos*
https://fccid.io/2AAIC00011

Fictiv, Swetha Sriram, *Anki Vector Robot Teardown*, 2019 Aug 6

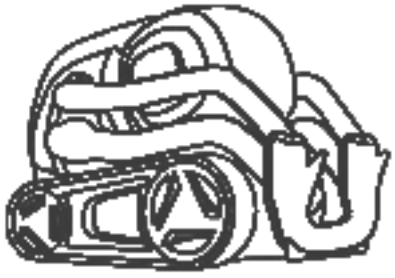
https://www.fictiv.com/blog/anki-vector-robot-teardown

Kinvert, *Anki Vector Customer Care Info Screen (CCIS)*
https://www.kinvert.com/anki-vector-customer-care-info-screen-ccis/

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

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
- SERVERS. This appendix provides the servers that the Anki Vector and App contacts
- PHRASES. This appendix reproduces the phrases that the Vector keys off of.
- FEATURES. This appendix enumerates the Vector OS "features" that can be enabled and disabled.
- FILE SYSTEM. This appendix lists the key files that are baked into the system.
- PLEO. This appendix gives a brief overview of the Pleo animatronic dinosaur.



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

APPENDIX A

Abbreviations, Acronyms, Glossary

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
EMR	*unknown*
ESD	electro-static discharge
ESN	electronic serial number
FBS	flat buffers
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
MSRP	manufacturer's suggest retail price
OLED	organic light-emitting diode display

Table 43: *Common acronyms and abbreviations*

OTA	over the air updates
PMIC	power management IC
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
TTS	text to speech
UART	universal asynchronous receiver/transmitter
USB	universal serial bus
UUID	universally unique identifier
vic	short Victor (Vector's working name)

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

Table 44: *Glossary of common terms and phrases*

rapidly-expanding random tree	A path-finding algorithm
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.
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 45: *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;
ARM NN	ARM's neural network support https://github.com/ARM-software/armnn
AudioKinetic Wwise[8]	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
Google FlatBuffers	Google FlatBuffers is used to encode the animation data structures https://github.com/google/flatbuffers

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

GNU C Compiler (gcc)	GCC version 4.9.3 was used to compile the kernel
golang	Go is used on the server applications, and (reported) some of Vectors internal software.
Google RPC (gRPC)	A "remote procedure call" standard, that allows mobile apps and the python SDK 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[9]	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

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

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/
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 46: *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.

20. CUBE SERVICES

times and other feature parameters:

Service	UUID[10]	Description & Notes
Device Info Service[11]	$180A_{16}$	Provides device and unit specific info – it's manufacturer, model number, hardware and firmware versions
Generic Access Profile[12]	1800_{16}	The device name, and preferred connection parameters
Generic Attribute Transport[13]	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 47: *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 48: *The Cube's Device info settings*

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

[11] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.device_information.xml

[12] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_access.xml

[13] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_attribute.xml

20.1. 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 49:** 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

21. VECTOR SERVICES SERVICE

times and other feature parameters:

Service	UUID[14]	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 50:** 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 51:** The Vector's Device info settings*

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

21.1. VECTORS SERIAL SERVICE

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

Table 52: *Vector's serial service characteristics*

APPENDIX D

Servers & Data Schema

This Appendix describes the servers that Vector contacts[15]

Server	Description & Notes
chipper.api.anki.com:443	The speech recognition engine lives here
conncheck.global.anki-services.com/ok	Used to check to see if it can connect to Anki
jdocs.api.anki.com:443	Storage of some sort of data. Name, faces, prefs?
token.api.anki.com:443	Used to get the API certificate.[16]
https://anki.sp.backtrace.io:6098	Vector posts crashes to this server
https://sqs.us-west-2.amazonaws.com/792379844846/DasProd-dasprodSqs-1845FTIME3RHN	This is used to synchronize with data analytics services.
https://ota.global.anki-services.com/vic/prod/	Where Vector checks for updates
https://ota.global.anki-dev-services.com/vic/rc/lo8awreh23498sf/	For the Developer branch
amazon.com/code	Where does the visual go?

Table 53: *The servers that Vector contacts.*

The servers that the mobile app contacts:

TBD

[15] Todo: sync up with info at: https://github.com/anki-community/vector-archive
[16] XYZ had a write up, reference that.

APPENDIX E

Phrases and their Intent

This Appendix maps the published phrases that Vector responds to and their intent:

Table 54: *The "Hey Vector" phrases*

Intent	Phrase
movement_backward	Back up
imperative_scold	Bad robot
	Be quiet
global_stop	Cancel the timer
	Change/set your eye color to [blue, green, lime, orange, purple, sapphire, teal, yellow].
check_timer	Check the timer
imperative_come	Come here
imperative_dance	Dance.
play_popawheelie	Do a wheelstand
imperative_fetchcube	Fetch your cube
imperative_findcube	Find your cube
play_fistbump	Fist Bump
play_fistbump	Give me a Fist Bump
movement_backward	Go backward
explore_start	Go explore
movement_forward	Go forward.
movement_turnleft	Go left
movement_turnright	Go right
	Go to sleep
	Go to your charger
	Good afternoon
greeting_goodbye	Goodbye
	Good evening
	Good night

greeting_goodmorning	Good morning
imperative_praise	Good robot
seasonal_happyholidays	Happy Holidays
seasonal_happynewyear	Happy New Year
greeting_hello	Hello
	He’s behind you
character_age	How old are you
imperative_abuse	I hate you.
knowledge_question	I have a question …
imperative_love	I love you.
imperative_apology	I’m sorry.
play_blackjack	Let’s play Blackjack
	Listen to music
imperative_lookatme	Look at me
	Look behind you
	My name is [Your Name]
imperative_negative	No
play_anygame	Play a game
play_anytrick	Play a trick
play_blackjack	Play Blackjack
play_pickupcube	Pick up your cube.
play_popawheelie	Pop a wheelie.
play_rollcube	Roll your Cube
	Run
set_timer	Set a timer for [length of time]
explore_start	Start Exploring
	Stop Exploring
global_stop	Stop the timer
take_a_photo	Take a picture of [me/us]
take_a_photo	Take a picture
take_a_photo	Take a selfie
movement_turnaround	Turn around
movement_turnleft	Turn left
movement_turnright	Turn right
imperative_volumelevel	Volume [number].
imperative_volumedown	Volume down
imperative_volumeup	Volume up.

	Volume maximum
names_ask	What's my name?
weather_response	What's the weather in [City Name]?
weather_response	What's the weather report?
show_clock	What time is it?
imperative_affirmative	Yes

Note: Vector's NLP server doesn't recognize "home" ..

Questions

Table 55: *The Vector questions phrases*

Subject	**Example Phrase**
Current conversion	What's 1000 Yen in US Dollars?
Flight status	What is the status of American Airlines Flight 100?
Equation solver	What is the square root of 144?
General knowledge	What is the tallest building?
places	What is the distance between London and New York?
People	Who is Jarvis?
Nutrition	How many calories are in an avocado?
Sports	Who won the World Series?
Stock market	How is the stock market?
Time zone	What time is it in Hong Kong?
Unit conversion	How fast is a knot?
Word definition	What is the definition of Artificial Intelligence?

Some of them are internal strings, some are hardcoded values

APPENDIX G

File system

This Appendix describes files systems

As the Vector uses the Android bootloader, it reuses – or at least reserves – many of the Android partitions[17] and file systems. Many are probably not used. Quotes are from Android documentation.

The file system table tells use where they are stored in the partitions, and whether they are non volatile.

***Table 56:** The file system mount table*

Mount point	Partition name	Description & Notes
/	BOOT_A	The primary linux kernel and initramfs
/data[18]	USERDATA	The data created for the specific robot (and user) that customizes it. A factory reset wipes out this user data. This portion of the file system is encrypted using "Linux Unified Key Setup" (LUKS).
/firmware	MODEM	The firmware for the WiFi/Bluetooth radio
/factory	OEM	Customizations, such as bootloader property values. … Or the factory recovery?
/persist	PERSIST	Device specific "data which shouldn't be changed after the device is shipped, e.g. DRM related files, sensor reg file (sns.reg) and calibration data of chips; wifi, bluetooth, camera etc."
/media/ram /run /var/volatile /dev/sm		Internal temporary file systems; holds temporary files, interprocess communication

The partition table found on the Vector:

***Table 57:** The partition table*

Partition name	Size	Description & Notes
ABOOT	1 MB	The application bootloader, which may load the kernel, recovery, or fastboot.
ABOOTBAK	1 MB	The application bootloader, which may load the kernel, recovery, or fastboot.
BOOT_A	32 MB	The primary linux kernel and initramfs.
BOOT_B	32 MB	The backup linux kernel and initramfs.
CONFIG	512 KB	Configuration of TBD.
DDR	32 KB	Configuration of the DDR RAM.
DEVINFO	1 MB	"device information including: is_unlocked (aboot), is_tampered, is_verified, charger_screen_enabled, display_panel, bootloader_version, radio_version etc.

[17] https://forum.xda-developers.com/android/general/info-android-device-partitions-basic-t3586565
[18] This is mounted by "mount-data.service' The file has a lot of information on how it unbricks

		Contents of this partition are displayed by "fastboot oem device-info" command in human readable format. Before loading boot.img or recovery.img, bootloader verifies the locked state from this partition."
EMR	16 MB	????
FSC	1KB	"Modem FileSystem Cookies"
FSG	1.5 MB	Golden backup copy of MODEMST1, used to restore it in the event of error
KEYSTORE	512 KB	"Related to [USERDATA] Full Disk Encryption (FDE)"
MISC	1MB	"a tiny partition used by recovery to communicate with bootloader store away some information about what it's doing in case the device is restarted while the OTA package is being applied. It is a boot mode selector used to pass data among various stages of the boot chain (boot into recovery mode, fastboot etc.). e.g. if it is empty (all zero), system boots normally. If it contains recovery mode selector, system boots into recovery mode."
MODEM	64 MB	Binary "blob" for the WiFi/Bluetooth radio firmware
MODEMST1	1.5MB	Binary "blob" for the WiFi/Bluetooth radio firmware
MODEMST1	1.5MB	Binary "blob" for the WiFi/Bluetooth radio firmware
OEM	16MB	Customizations, such as bootloader property values.
PAD	1MB	"related to OEM"
PERSIST	64MB	Device specific "data which shouldn't be changed after the device is shipped, e.g. DRM related files, sensor reg file (sns.reg) and calibration data of chips; wifi, bluetooth, camera etc."
RECOVERY	32 MB	An alternate partition holding systems applications and libraries that let the application boot into a mode that can download a new system. Often used to wipe out the updates.
RECOVERYFS	640 MB	An alternate partition holding systems applications and libraries that let the application boot into a mode that can download a new system. Often used to wipe out the updates.
RPM	512KB	The primary for resource power management[?]
RPMBAK	512KB	The backup for resource power management[?]
SBL1	512KB	Secondary bootloader. Responsible for loading ABOOT; may also have an "Emergency" download mode.
SBL1BAK	512KB	Secondary bootloader. Responsible for loading ABOOT; may also have an "Emergency" download mode.
SEC	16KB	The secure boot fuse settings, OEM settings, signed-bootloader stuff
SSD	8KB	"Secure software download" for secure storage, encrypted RSA keys, etc
SYSTEM_A	896MB	The (primary) systems applications and libraries with application specific code.
SYSTEM_B	896MB	The bakcup systems applications and libraries with application specific code.
SWITCHBOARD	16 MB	????
TZ	768KB	The TrustZone of key-value pairs, encrypted by the hardware and other keys
TZBAK	768KB	The TrustZone of key-value pairs, encrypted by the hardware and other keys
USERDATA	768MB	The data created for the specific robot (and user) that customizes it. A factory reset wipes out this user data. This partition is encrypted using "Linux Unified Key Setup" (LUKS).

The files employed in the Vector binaries:

Table 58: *Files*

File	Description
/anki/etc/version	
/data/data/com.anki.victor	
/data/data/com.anki.victor/cache/crashDumps	
/data/etc/localtime	The time zone
/data/misc/bluetooth/abtd.socket	The IPC socket interface to Bluetooth LE
/data/unbrick	
/dev/block/bootdevice/by-name/emr	
/dev/spidev0.0	The SPI channel to communicate with the IMU
*/dev/socket/_anim_robot_server_*	The IPC socket with Vector's animation controller
*/dev/socket/_engine_gateway_server_*	The IPC socket interface to Vector's Gateway [TBD] server
*/dev/socket/_engine_gateway_proto_server_*	The IPC socket interface to Vector's Gateway [TBD] server
*/dev/socket/_engine_switch_server_*	The IPC socket interface to Vector's Switchbox [TBD] server
/dev/ttyHS0	Console log? (but then, where is the connection to the base-board?)
/factory/cloud/something.pem	
/proc/sys/kernel/random/boot_id	A random identifier, created each boot
/sys/devices/system/cpu/possible[19] */sys/devices/system/cpu/present*	The number of CPUs and whether they can be used.
/data/data/com.anki.victor/cache/crashDumps	Parameters and values specific to the ST LIS2DH accelerometer
/run/anki-crash-log	
/run/das_allow_upload	
/run/fake-hwclock-cmd[20]	Sets the fake time to the time file (Vector doesn't have a clock)
/run/fault_code	
/tmp/data_cleared	
/tmp/vision/neural_nets	

[19] https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-devices-system-cpu
[20] https://manpages.debian.org/jessie/fake-hwclock/fake-hwclock.8.en.html

APPENDIX H

Pleo

The Pleo, sold in 2007 –a decade prior to Vector – has many similarities. The Pleo was a software skinned animatronic baby dinosaur created by Caleb Chung, John Sosuka and their team at Ugobe. Ugobe went bankrupt in 2009, and the rights were bought by Innvo Labs which introduced a second generation in 2010. This appendix is mostly adapted from the Wikipedia article and reference manual.

Sensing for interacting with a person

- Two microphones, could do beat detection allowing Pleo to dance to music. The second generation (2010) could localize the sound and turn towards the source.
- 12 touch sensors (head, chin, shoulders, back, feet) to detect when petted,

Environmental sensors

- Camera-based vision system (for light detection and navigation). The first generation treated the image as gray-scale, the second generation could recognize colors and patterns.
- Four ground foot sensors to detect the ground. The second generation could prevent falling by detecting drop-offs
- Fourteen force-feedback sensors, one per joint
- Orientation tilt sensor for body position
- Infrared mouth sensor for object detection into mouth, in the first generation. The second generation could sense accessories with an RFID system.
- Infrared detection of objects
- Two-way infrared communication with other Pleos
- The second generation include a temperature sensor

Annuciators and Actuators

- 2 speakers, to give it sounds
- 14 motors
- Steel wires to move the neck and tail (these tended to break in the first generation)

The processing

- Atmel ARM7 microprocessor was the main processor.
- An NXP ARM7 processor handle the camera system, audio input
- Low-level motor control was handled by four 8-bit processors

A developers kit – originally intended to be released at the same time as the first Pleo – was released ~2010. The design included a virtual machine intended to allow "for user programming of new behaviors."[21]

21.2. SALES

Pleo's original MSRP was $350, "the wholesale cost of Pleo was $195, and the cost to manufacture each one was $140" sold ~100,000 units, ~$20 million in sales[22]

The second generation (Pleo Reborn) had an MSRP of $469

21.3. RESOURCES

Wikipedia article. https://en.wikipedia.org/wiki/Pleo

iFixit's teardown. https://www.ifixit.com/Teardown/Pleo+Teardown/597

Ugobe, *Pleo Monitor,* Rev 1.1, 2008 Aug 18

Ugobe, *Pleo Programming Guide,* Rev 2, 2008 Aug 15

[21] https://news.ycombinator.com/item?id=17755596
[22] https://www.idahostatesman.com/news/business/article59599691.html
https://www.robotshop.com/community/blog/show/the-rise-and-fall-of-pleo-a-fairwell-lecture-by-john-sosoka-former-cto-of-ugobeJohn Sosoka