TECHNICAL DEEP DIVE

# *Anki Vector*

## A LOVE LETTER TO THE LITTLE DUDE


AUTHOR RANDALL MAAS


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

*drawing 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

# Table of Contents

# Figures

# Tables

## Examples



## Equations

# 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 hardware teardowns from others; by a team of people (Project Victor) analyzing the released software; and by experience with the parts, and the functional areas.

### 1.1. VERSION(S)

The software analyzed here is mostly version 1.5 and version 1.6 of Vector. There are incremental differences with each version; I have not always described the places that only apply to a specific version. Version 1.6 was a release rushed to customers as Anki ceased operation. This release includes more software elements that are unused, but are nonetheless telling.

### 1.2. CUSTOMIZATION AND PATCHING

What can be customized – or patched – in Vector?

- The software in the main processor may be customizable; that will be dicussed in many areas of the rest of the document
- The base-board firmware is field updatable, and will take expertise to construct updates.
- The cube firmware can be updated, but that appears to be the hardwest to change, and not likely to be useful.

## 2. ORGANIZATION OF THIS DOCUMENT

- PREFACE. This introduction describes the organization of the chapters and appendices.
- CHAPTER 1: OVERVIEW OF VECTOR'S ARCHITECTURE. Introduces the overall design of the Anki Vector robot.

PART I: ELECTRICAL DESIGN. This part provides an overview of the design of the electronics in Vector and his accessories:

- CHAPTER 2: VECTOR'S ELECTRICAL DESIGN. A detailed look at the electrical design of Vector.
- CHAPTER 3: ACCESSORY ELECTRICAL DESIGN. A look at the electrical design of Vector's accessories.

PART II: BASIC OPERATION. This part provides an overview of Vector's software design.

- CHAPTER 4: ARCHITECTURE. A detailed look at Vector's overall software architecture.
- CHAPTER 5: STARTUP. A detailed look at Vector's startup, and shutdown processes

- Chapter 6: Power Management. A detailed look at Vector's architecture for battery monitoring, changing and other power management.
- Chapter 7: Button & Touch Input and Output LEDs
- Chapter 9: Audio Input
- Chapter 10: Motion Sensing

Part III: Communication. This part provides details of Vector's communication protocols. These chapters describe structure communication, the information that is exchange, its encoding, and the sequences needed to accomplish tasks. Other chapters will delve into the functional design that the communication provides interface to.

- Chapter 11: Communication. A look at Vector's communication stack.
- Chapter 12: Bluetooth LE. The Bluetooth LE protocol that Vector responds to.
- Chapter 13: SDK Protocol. The HTTPS protocol that Vector responds to.
- Chapter 14: Cloud. A look at how Vector syncs with remote services.

Part IV: Advanced Functions.

- Chapter 15: Image Processing. A look at how Vector vision system
- Chapter 16: Mapping & Navigation. A look at Vector's mapping and navigation systems.
- Chapter 17: Accessories. A look at Vector's home (charging station), companion cube and custom objects.

Part IV: Animation

- Chapter 18: Audio Production. A look at how Vector's sound effects and how he speaks
- Chapter 19: Display & Procedural Face. At look at how Vector's moves.
- Chapter 20: Motion Control. At look at how Vector's moves.
- Chapter 21: Animations. A look at Vector's scripted motions and sounds.

Part IV: Maintenance. This part describes items that are not Vector's primary function; they are practical items to support Vector's operation.

- Chapter 22: Settings, Preferences, Features and Statistics. A look at how Vector syncs with remote servers
- Chapter 23: Software Updates. How Vector's software updates are applied.
- Chapter 24: Diagnostics. The diagnostic support built into Vector, including logging and usage statistics.

References and Resources. This provides further reading and referenced documents.

Appendices: The appendices provide extra material supplemental to the main narrative. These include tables of information, numbers and keys.

- 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: ALEXA MODULES. This appendix describes the modules used by the Alexa client
- APPENDIX D: FAULT AND STATUS CODES. This appendix provides describes the system fault codes, and update status codes.
- APPENDIX E: FILE SYSTEM. This appendix lists the key files that are baked into the system.
- APPENDIX F: BLUETOOTH LE PROTOCOLS. This appendix provides information on the Bluetooth LE interfaces to the companion Cube, and to Anki Vector.
- APPENDIX G: SERVERS. This appendix provides the servers that the Anki Vector and App contacts.
- APPENDIX H: FEATURES. This appendix enumerates the Vector OS "features" that can be enabled and disabled.
- APPENDIX I: PHRASES. This appendix reproduces the phrases that Vector keys off of.
- APPENDIX J: DAS EVENTS. This appendix describes the identified DAS events
- APPENDIX K: PLEO. This appendix gives a brief overview of the Pleo animatronic dinosaur, an antecedent with many similarities.

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

## 2.1. ORDER OF DEVELOPMENT

A word on the order of development; the chapters are grouped in sections of related levels of functionality and (usually) abstraction.

Most chapters will description a vertical slice or stack of the software. The higher levels will discuss features and interactions with other subsystems that have not been discussed in detail yet. For instance, the section on the basic operation of Vectors hardware includes layers that link to the behavior and communication well ahead of those portions. Just assume that you'll have to flip forward and backward from time to time.

The communication interface is held to its own section with the relevant interactions, commands, structures and so on.

CHAPTER 1

# Overview of Vector

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

- Overview
- Privacy and Security
- Ancestry: Cozmo
- Alexa Built-in

## 3. OVERVIEW

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

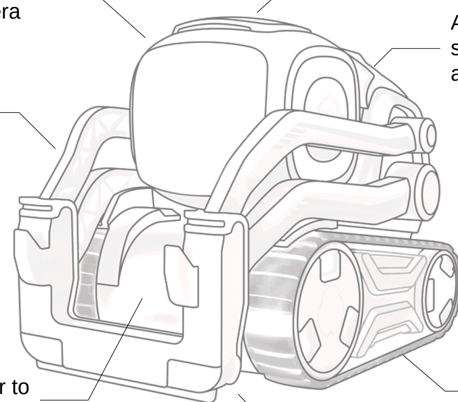



***Figure 1:** Vector's main features*

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 expensive, 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/annunciators:

- Segmented lights on Vector's 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 use 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 *inertial measurement unit* (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 at the edge of his area and to following lines when he is reversing onto his charger.

His internal sensing includes:

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

His other articulation & actuators are:

- 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.

[4] and possibly a pat on his head?

- Internally RS-232 (CMOS levels) and USB

Motion control

- At the lowest level can control each of the motors speed, degree of rotation, etc. This allows Vector to make quick actions.
- Combined with the internal sensing, he can drive in a straight line and turn very tightly.
- Driving is done using a skid-steering, kinematic model
- To do all this, the motion control takes in feedback from the motor encoder, IMU-gyroscope. May also use the image processing for SLAM-based orientation and movement.

Guidance, path planning

- Vector plans a route to his goals – if he knows where his goal is – along a path free of obstacles; he adapts, moving around in changing conditions.
- A*, Rapidly-Expanding Random Tree (RRT), D*-lite
- Paths are represented as arcs, line segments, and turn points

Mapping and Navigation:

- Maps are built using *simultaneous location and mapping* (SLAM) algorithms, using the camera and IMU gyroscope movement tracking, time of flight sensor to measure distances, and particle system algorithms to fill in the gaps.
- The maps are represented uses quad-tree (position, pose)

Behaviour system:

- Variety of behaviors animations
- Goals, linking up to the guidance system to accomplish them
- A simple emotion model to drive selection of behaviours

Emotion model. Dimensions to emotional state

- Happy (also referred to as his default state)
- Confident
- Social
- Stimulated

Vision. This is one of Anki's hallmark: they used vision where others used beacons. For instance, iRobot has a set of IR beacons to keep the robots of out areas, and to guide it to the dock. Mint has an IR beacon that the mint robots use to navigate and drive in straight lines. Although Vector's companion cube is powered, this is not used for localization. It has markers that are visually recognized by Vector.

- Illumination sensing
- Motion sensing
- Links to Navigation system for mapping, (SLAM etc)
- Recognizing marker symbols in his environment

- Detecting faces and gaze detection allows him to maintain eye contact

# 4. PRIVACY AND SECURITY

Vector's design includes a well thought out system to protect privacy. This approach protects the following from strangers gaining access:

- Photos taken by Vector
- The image stream from the camera
- The audio stream from the microphone — if it had been finished being implemented
- Information about the owner
- Control of the robot's movement, speech & sound, display, etc.

Vector's software is protected from being altered in a way that would impair its ability to secure the above.

# 5. COZMO

We shouldn't discuss Vector without mentioning the prior generation. Vector's body is based heavily on Cozmo; the mechanical refinements and differences are relatively small. Vector's software architecture also borrows from Cozmo and extends it greatly. Many of Vector's behaviours, senses, and functions were first implemented in Cozmo (and/or in the smartphone application). One notable difference is that Cozmo did not include a microphone.

Cozmo includes a wide variety of games, behaviours, and ~940 animation scripts. Cozmo's engine is reported to be "about 1.8 million lines of code, the AI, computer vision, path planning, everything."[5] This number should be discounted somewhat, as it likely includes many large 3rd party modules… Nonetheless, it represents the scale of work to migrate Cozmo's code base for reuse in Vector.

Not all of Cozmo's functionality was ported to Vector at one time. Instead, key features and behaviours were incrementally brought to Vector in its regular software updates. It is likely the intent was to follow-up with much more in future updates (and perhaps on a faster schedule than they were able to deliver).

# 6. ALEXA INTEGRATION

Vector includes Amazon Alexa functionality, but it is not intimately integrated. Vector only acts like an Echo Dot. By using the key word "Alexa," Vector will suppress his activity, face and speaking, and the Alexa "echo dot" functionality takes over. Vector has no awareness of Alexa's to-do list, reminders, messages, alarms, notifications, question-and-answers, and vice-versa.

The most likely reason for including Alexa is the times: everything had to include Alexa to be hip, or there would be great outcry. Including Alexa may have also been intended to provide functionality and features that Anki couldn't, to gain experience with the features that Amazon provides, and (possibly) with the intent to more tightly integrate those features into Anki products while differentiating themselves in other areas.

[5] https://www.reddit.com/r/IAmA/comments/7c2b5k/were_the_founders_of_anki_a_robotics_and_ai/

Alexa clearly took a lot of effort to integrate, and a lot of resources:

> "[Alexa Voice Service] solutions for Alexa Built-in products required expensive application processor-based devices with >50MB memory running on Linux or Android"[6]

Alexa's software resources consume as much space Vector's main software. And the software is not power efficient. Even casual use of Alexa noticeably reduces battery life, and (anecdotally) increases the processor temperature.

See Appendix C for a list for a list of the Alexa modules.

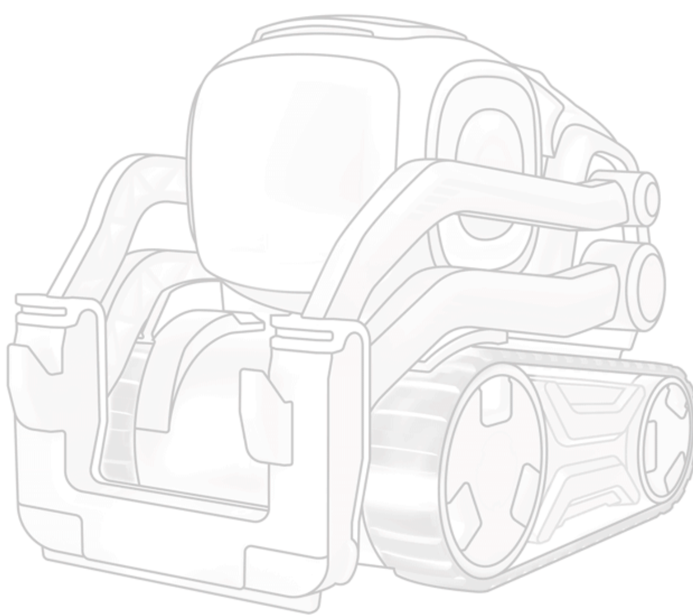
[6] https://aws.amazon.com/blogs/iot/introducing-alexa-voice-service-integration-for-aws-iot-core/
Alexa's SDK and services have continued to evolve. New Alexa SDKs allow simpler processors and smaller code by acting as little more than a remote microphone.

## 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 2

# 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

## 7. 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.

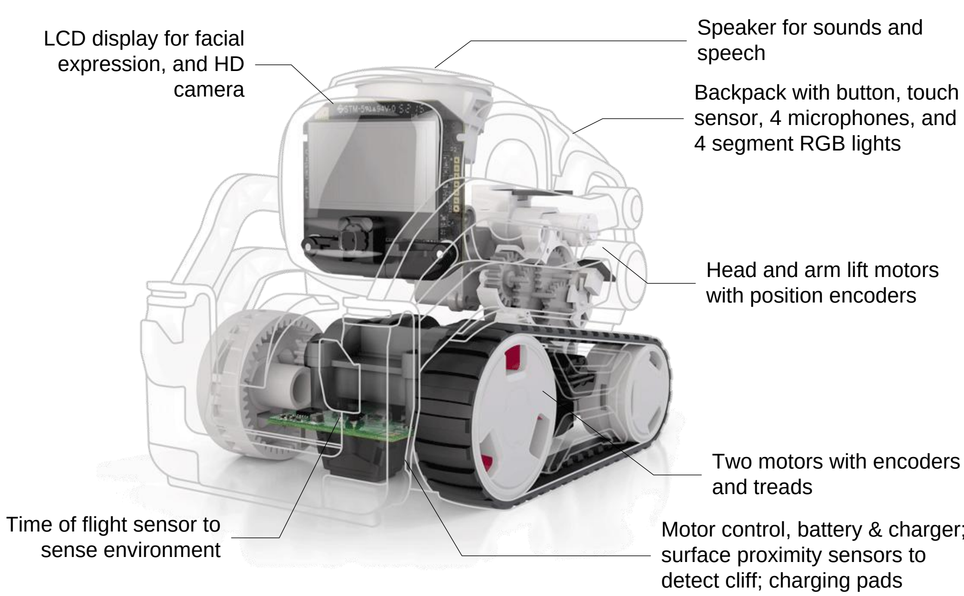



***Figure 2:** Vector's main elements*

Vector's functional elements are:

*Table 1: Vector's main elements*

| 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 microphone(s) to listen to commands and ambient activity level. Employs beam forming to localize sounds. |
| motors & encoders | There are four motors each with magnetic 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, and to follow lines. |
| 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. |

Vector has 6 circuit boards

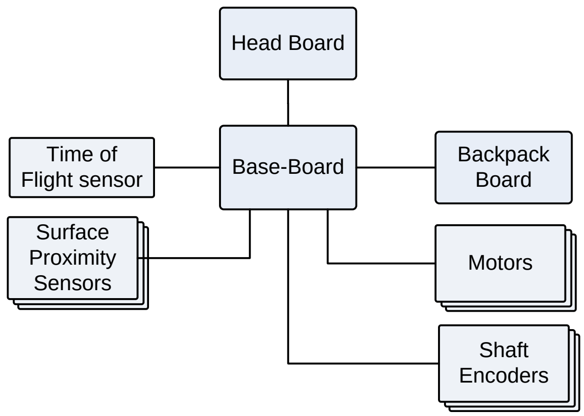



***Figure 3:*** *Circuit board topology*

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

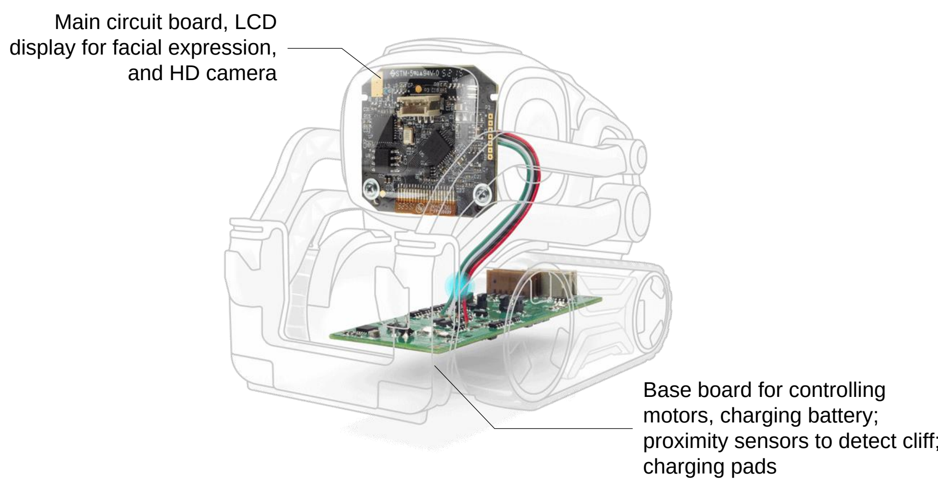



***Figure 4:*** *Vector's main microcontroller circuit boards*

The table below summarizes the boards:

***Table 2:*** *Vector's circuit 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 magnetic quadrature 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 Vector's front. |

## 7.1. POWER SOURCE AND DISTRIBUTION TREE

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

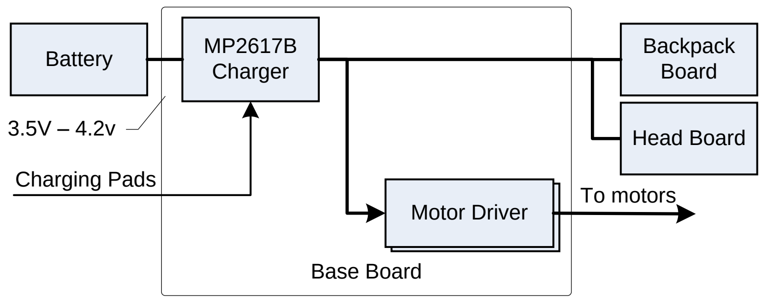



***Figure 5:*** *Power distribution*

The MP2617B 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 MP2617B 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 MP2617B to connect the battery to the rest of the system, powering it up.
- The MP2617B 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.

Excessive current demand – such as from a stalled motor – can trigger a system brown-out and shutdown.

# 8. THE BACKPACK BOARD

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

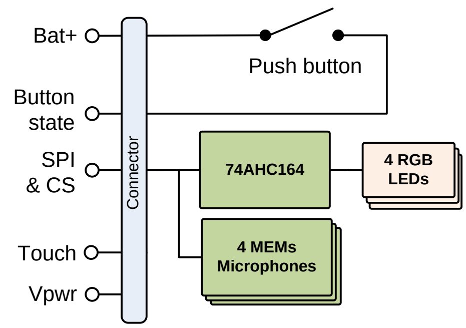



***Figure 6:*** *Backpack board block diagram*

The table below summarizes the functional elements of the backpack board:

***Table 3:*** *Backpack board functional elements*

| Elements | Description |
|---|---|
| 74AHC164 | A SPI-based GPIO expander. This is used to drive the RGB LEDs. |
| microphones | There are 4 internal MEMS microphone(s). The microphones are accessed via SPI, in an output only mode. These are designated MK1, MK2, MK3, MK4 |
| push button | A momentary push button is connected to the battery terminal, allowing a press to wake Vector, as well as signal the processor(s). |
| RGB LEDs | There are 4 RGB LEDs to make up a segmented display. Each segment can be illuminated individually but may share a colour configuration with its counterparts. |
| touch sensor | A touch-sensing wire (and passive components) |

## 8.1. 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 button
- (at least) Two chip selects
- A SPI-like set of clock, master-out-slave-in (MOSI) and *two* master-in-slave-out (MISO) signals

## 8.2. OPERATION

The touch sensor conditioning and sensing is handled by the base-board. The touch sense wire is merely an extension from the base-board through the backpack 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 theory of operation will be discussed further in the base-board section below.

The 74AHC164 serial-shift-register is used as a GPIO expander. It takes a chip select, clock signal and serial digital input, which are used to control up to 8 outputs. 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).

### 8.2.1 The LED controls

*Backpack LED control scheme*

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 may have a different colour 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.[7]
- 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).

[7] 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

# 9. THE BASE-BOARD

The base board is a battery charger, smart IO expander, and motor controller. It connects the battery to the rest of the system and is responsible for charging it. It is based on an STM32F030 which acts as second processor in the system.

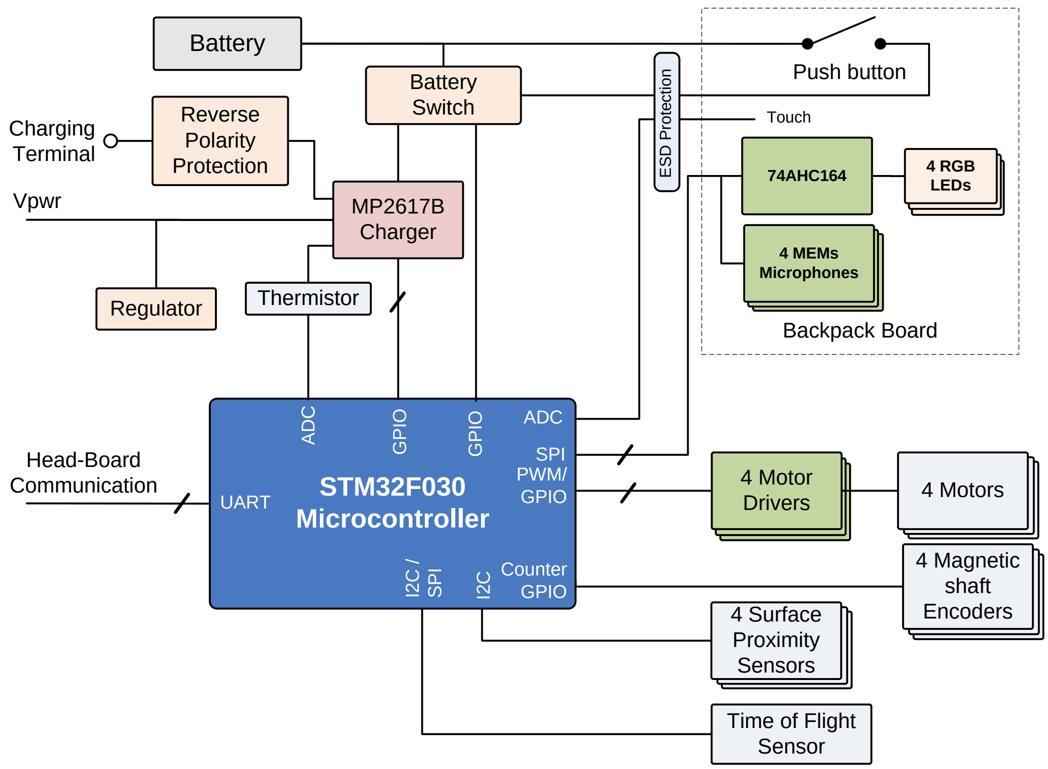



***Figure 7:*** *Base-board block diagram*

The functional elements of the base-board are:

***Table 4:*** *The base-board functional elements*

| **Element** | **Description** |
|---|---|
| battery | An internal, rechargeable battery pack (3.7v 320 mAh) |
| battery switch | Used to disconnect the battery to support off-mode (such as when stored) and to reconnect the battery with a button press. |
| charging pad | Two pads on the bottom are used to replenish the energy in the battery pack from the dock. |
| motor driver | There are four motor drivers, based on an H-bridge design. This allows a motor to be driven forward and backward. |
| motors | There are four motors with 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. |
| MP2617B charger | The Monolithic Power Systems MP2617B serves as the battery charger. It provides a state of charge to the microcontroller. |
| magnetic shaft encoder | A magnetic quadrature encoder – two-hall switches, in conjunction with a magnetic disc on a motor's shaft –is used to measure the amount a shaft has turned, and its speed. |
| regulator | A 3.3v used to supply power to the microcontroller and logical components. |
| reverse polarity protection | Protects the circuitry from energy being applied to the charging pads in reverse polarity, such |

| | |
|---|---|
| | as putting Vector onto the charging pads in reverse. |
| STM32F030 microcontroller | The “brains” of the baseboard, used to drive the motors, and RGB LEDs; to sample the microphones, time of flight sensor, proximity sensor, temperature, and the touch sense;, and monitoring the battery charge state. It communicates with the head-board. |
| 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 and to follow lines. |
| thermistor[8] | A temperature sense resistor used measure the battery pack temperature; it is used to prevent overheating during recharge. |
| VL53L1 time of flight sensor | A ST Microelectronics VL53L1 time of flight sensor is used to measure distance to objects in front of Vector. This sensor is connected by $I^2C$. |

## 9.1. POWER MANAGEMENT

The battery charging is based on a MP2617B IC, which also provides some protection functions. There is no Coulomb counter; the state of charge is based solely on the battery voltage.

### 9.1.1 Battery pack

Vector’s single-cell lithium battery is connected to the baseboard and laid on top of the PCBA. The battery is not removable. The battery label has it as a 3.7v 320mAh pack. It is rechargeable. The pack is not a “smart” battery – it only has positive and negative leads but lacks an onboard temperature sensor or BMS.

### 9.1.2 Protections

The charging pads have reverse polarity protection.

The MP2617B has an over-current cut off. If the current exceeds ~5A (4-6A), the battery will be disconnected from the system bus. Such a high-current indicates a short. There is no fuse.

The MP2617B has a low voltage cut off. If the battery voltage drops below ~2.4 (2.2-2.7V) the battery will be disconnected from the system bus (TBD) until the battery voltage rises above ~2.6V (2.4-2.8V).

The MP2617B has a temperature sense. If the temperature exceeds a threshold, charging is paused until the battery cools. The temperature sense is not on the battery. It is likely on the circuit board, or possibly top of the battery retention.

[8] Not identified. The customer service screen does show a battery pack temperature, indicating that this is reported.

### 9.1.3 Battery connect/disconnect

To preserve the battery there is a need to isolate the battery from the rest of the system when in an off state. If there is minute current draw, the battery will irreversibly deplete while in storage even before the first sale. This constraint shapes the battery disconnect-reconnect logic. The schematic below shows one way to do this:

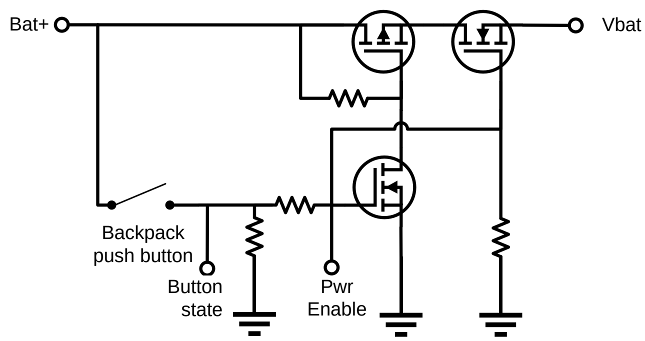



***Figure 8:** A representative battery connect switch*

Two MOSFETS (a PFET and NFET)[9] act as a switch. These are in a single package, the DMC2038LVT. (This part is also used in the motor drivers.)

- When the system is in an off state, the MOSFETs are kept in an off state with biasing resistors. The PFET's gate is biased high with a resistor. The NFET gate is biased low, to ground. There is no current flow. Two MOSFETS are needed due to internal body diodes. The PFET body diode would allow current to flow from the battery (from the source to the drain). However, this current is blocked by the NFET body diode, which has a different polarity
- The push button can wake the system. When the button is closed, the battery terminal (Bat+) is connected to the gate of the NFET, turning it on. A second NFET is also energized, pulling the PFET gate to ground, turning it on as well. When the button is open, Bat+ is not connected to anything, so there is no leakage path draining the battery.
- To keep the system energized when the button is open, the STM32F030 MCU must drive the Pwr Enable line high, which has the same effect as the button closed. The gate threshold voltage is 1V, well within the GPIO range of the MCU.
- The MCU can de-energize the system by pulling Pwr Enable line low. The switches will open, disconnect the battery.
- The MCU needs to be able to sense the state of the button while Pwr Enable is pulled high. The MCU can do this by sampling the Button State signal. This signal is isolated from from Pwr Enable by a large resistor and pulled to ground by smaller resistor. This biases the signal to ground while the button is open.

This circuit also provides reverse polarity protection. It will not close the switch if the battery is connected backwards.

### 9.1.4 Charging

*charging station pads*

The charging station pads are connected to a MP2617B charger IC thru a reverse polarity protection circuit. The reverse polarity protection[10] is a DMG2305UX PFET in a diode

[9] Q11 and/or Q12
[10] Q14

configuration. This approach has much lower losses than using an equivalent diode.

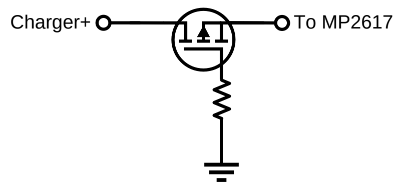



***Figure 9:** A representative PFET based reversed polarity protection*

The MP2617B internally switches the charger input voltage to supply the system with power, and to begin charging the battery. This allows the charger to power the system even when the battery is depleted, or disconnected.

*supplying power from the charging station*

The presence of the dock power, and the state of MP2617B (charging or not) are signaled to the microcontroller.

The charger goes through different states as it charges the battery. Each state pulls a different amount of current from the charging pads and treats the battery differently.

*charging states*

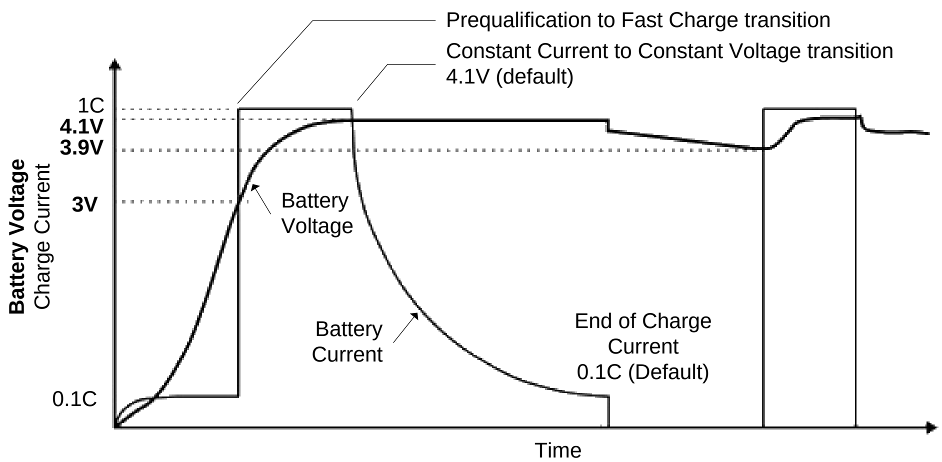



***Figure 10:** Charging profile* (adapted from Texas Instruments)

The basic idea is that the charger first applies a low current to the battery to bring it up to a threshold; this is called *prequalification* in the diagram. Then it applies a high current, call *constant current*. Once the battery voltage has risen to a threshold, the charger switches to *constant voltage*, and the current into the battery tapers off. I refer to the data sheet for more detail.

*constant current constant voltage*

The MP2617B measures the battery temperature using a thermistor. If the temperature exceeds a threshold, charging is paused until the battery cools. The microcontroller also samples this temperature.

The MP2617B supports limiting the input current, to accommodate the capabilities of external USB power converts. There are four different possible levels that the IC may be configured for: 2A is the default limit, 450mA to support USB2.0 limits, 825mA to support USB3.0 limits, and a custom limit that can be set by resistors. The input limit appears to be set for either default (up to ~2A input), or a programmable input.

*input current limits*

*Commentary.* In my testing, using a USB battery pack charging pulls up to 1A during the constant current, then falls off to 100mA-200mA during constant voltage, depending on the

*Is the charger damaging the battery?*

head-board's processing load. Stepped down to the ~4V battery the applied current at peak is approximately 1A.[11] This seems far too high.

Battery cells are normally charged at no more than a "1C" rate – in this case, the battery maximum charge rate should be 320mA at max. The IC data sheet supports a charging rate up to 2A.

My speculation is that, intentionally or unintentionally, the charger is configured for the default input limit of 2A and supports a faster charge. It is possible that the impact to battery life was considered low. My analysis could be wrong. As a preventative measure, I have a current limiter between my USB power adapter and Vector's charging dock.[12]

### 9.1.5 Brown-out

The motor stall current is enough to cause Vector to brown-out and shut down unexpectedly. This indicates two possible mechanisms:

*motor stall & brown out effects*

- If the system browns out the STM32F030, the MCU will no longer hold the power switch closed, and the system power will be disconnected.
- If the current exceeds a threshold, the MP2617B will disconnect power to the system. This threshold is very high – ~5A – and is unlikely to ever be encountered in operation.

*Commentary:* It may be interesting to modify either the MCU's Vdd to have a larger retaining capacitor, or to add a current limiting mechanism for the motors, such as an inline resistor.

## 9.2. ELECTRO-STATIC DISCHARGE (ESD) PROTECTION

The base-board employs a Vishay GMF05, TVS diode (U4) for electro-static discharge (ESD) protection, likely on the pushbutton and touch input.

## 9.3. STM32F030 MICROCONTROLLER

The base-board is controlled by a STM32F030C8 microcontroller (MCU). This processor essentially acts as a smart IO expander and motor controller.

The MCU's digital inputs:

- 8 hall-switches used in 4 quadrature encoders, one for each motor (left, right, head, lift)
- Momentary push button
- 4 IR proximity sensor used to detect cliffs and lines
- 2 charger state

The MCU's digital outputs:

- 4 motors enable
- 4 motors direction
- charger enable
- 3 chip selects

The MCU's analog inputs:

- Touch

[11] Other reports suggest up to 2As into the battery, possible with the use of high-power USB adapters intended to support tablet recharge.
[12] 1Ω on the USB power. I tried 1Ω -14Ω; these should have limited the current to 1A and 500mA respectively. Instead, Vector would only pull 40mA - 370mA; in many cases, not enough to charge.

- Battery voltage
- Temperature sensor (picks off the thermistor used by the MP2617)
- [Possibly] an current measurement

The communication:

- 2 SPI, to LED outputs, from microphones. Uses an SPI MCLK to clock out the state, and MOSI to send the state of that IO channel
- I2C for communication with the time of flight sensor
- UART, for communication with the head board

Note: The microcontroller does not have an external crystal[13] and uses an internal RC oscillator instead.

### 9.3.1 Manufacturing test connector

The base-board does include pads that appear to be intended for programming and test at manufacturing time.

### 9.3.2 Firmware updates

The firmware is referred to as "syscon". The microcontroller includes a boot loader, allowing the firmware to be updated by the head-board. The firmware can be updated in OTA software releases.

Future changes to the base-board firmware will require expertise. The STM32F030 firmware can by analysis of the syscon.dfu file (or be extracted with a ST-Link) and disassembled. Shy of recreating the firmware source codes, 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.

Emulation (such as QEMu-STM32) , ST-link ($25) and a development environment will be required to debug and modify the firmware initially. The development environment ranges from free to several thousand dollars, the later being the more productive tools.

## 9.4. SENSING

### 9.4.1 Time of Flight sensor

The MCU interfaces with a ST Microelectronics VL53L1 time of flight sensor, which can measure the distance to objects in front of vector. It "has a usable range 30mm to 1200mm away (max useful range closer to 300mm for Vector) with a field of view of 25 degrees." Anki SDK

These sensors work by timing how long it takes for a coded pulse to return. The time value is then converted to a distance. Items too close return the pulse faster than the sensor can measure. The measured distance is available to the microcontroller over $I^2C$.

### 9.4.2 Proximity sensing

Has 4 IR proximity sensors that are used to used to detect drops offs ("cliffs") and to follow lines. The exact model hasn't been identified, but the Everlight EAAPMST3923A2 is a typical proximity sensor. The sensor is an LED and IR detector pair. The sensor reports, via $I^2C$, the brightness sensed by the detector. This are often pulsed, to reject to sunlight; and use a configurable threshold to reduce sensitivity to ambient light.

---

[13] as far as I can see

### 9.4.3 Touch sensing

Anki SDK

The touch sensing works by alternating pulsing and sampling (with the ADC) the touch wire. The samples will vary "by various environmental factors such as whether the robot is on its charger, being held, humidity, etc."

## 9.5. MOTOR DRIVER AND CONTROL

Each motor driver is an H-bridge, allowing a brushed-DC motor to turn in either direction.

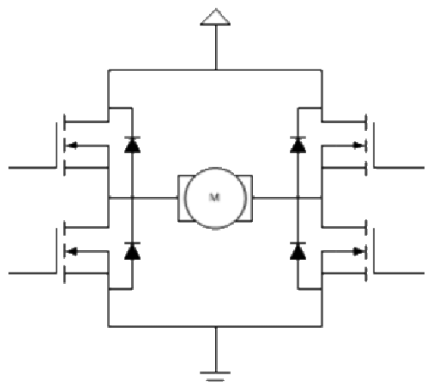



***Figure 11:** Motor driver H-bridge*

Each side of the H-bridge based on the DMC2038LVT, which has a P-FET and N-FET in each package. Two of these are needed for each motor.

The MCU (probably) independently controls the high side and low side to prevent shoot thru. This is done by delaying a period of time between turning off a FET and turning on a FET.

The motors can be controlled with a control loop that takes feedback from the optical encoder to represent speed and position.

## 9.6. COMMUNICATION

The base-board communicates with the head-board via RS-232 3.3V (3 Mbits/sec[14]). As the MCU does not have a crystal, there may be communication issues from clock drift at extreme temperatures; since Vector is intended for use at room temperature, the effect may be negligible.

The firmware can be updated over the serial communication by the head-board.

*SPI communication with 2 microphones simultaneously*

The communication with the backpack board is special. Two microphones are read at a time, using a shared SPI clock and chip select. The process can be:

1. The first chip select is asserted
2. A 16-bit SPI transfer is initiated on two SPI ports nearly simultaneously; the clock and data output (MOSI) on the second is ignored. This may be done carefully in code with as little as 1-instruction cycle skew.
    a. This transfer sends the state of the RGB LED's to the 74AHC164 chip
    b. The receiver accepts 16-bits each from the microphone 1 and 3.
3. After completion, the first chip select is de-asserted, and the second chip select is asserted.

[14] Value from analyzing the RAMPOST program. Melanie T measured it on an oscilloscope and estimated it to be 2Mbps.

4. A 16-bit SPI transfer is initiated on two SPI ports nearly simultaneously; the clock and data output (MOSI) on the second is ignored. This transfers 16-bits each from the microphone 2 and 4.

5. After completion, the second chip select is de-asserted

The microphones are sampled a rate of 15625 samples/sec.

# 10. THE HEAD-BOARD (THE MAIN PROCESSOR BOARD)

The head-board handles the display, playing sounds, communication, and all of Vector's real processing. It is powered by a quad-core Arm-A7 Qualcomm APQ8009 microprocessor. The processor also connects to Bluetooth LE and WiFi transceivers, an HD camera, LCD display, speakers and an IMU.

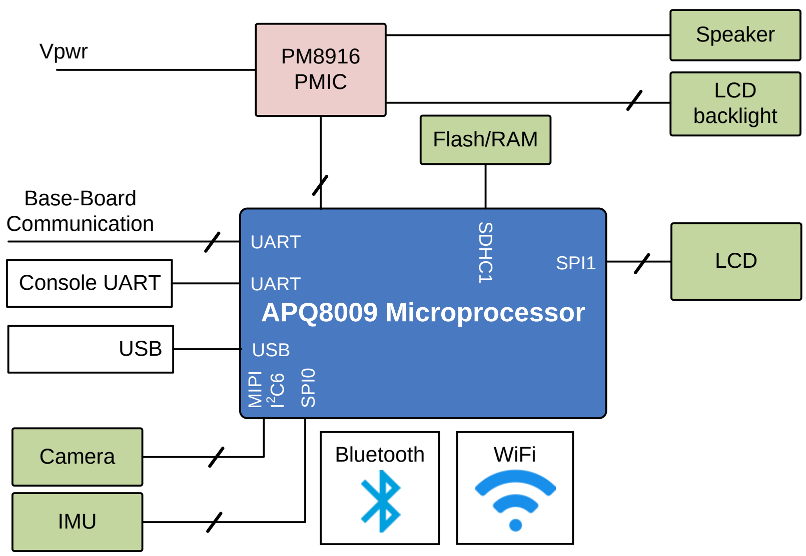



***Figure 12:** Head-board block diagram*

The head-board's functional elements are:

| Element | Description |
|---|---|
| Bluetooth LE transceiver | A Bluetooth LE transceiver is built into the package |
| camera | Vector uses a 720P camera to visualize his environment and recognize his human companions. |
| flash/RAM (eMMC) | Flash and RAM are provided by single external package, a Kingston 04EMCP04-NL3DM627 mixed memory chip with 4 GB flash and 512MB RAM. |
| inertial measurement unit (IMU) | The headboard includes a 6-axis IMU – gyroscope and accelerometer – used for navigation and motion control. |
| LCD backlight | There are two LEDs used to illuminate the LCD display. |
| 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. |
| microprocessor | The head-board is based on a Qualcomm APQ8009 (Snapdragon 212). The processor is a quad-core Arm A7 (32-bit) CPU. |
| power management IC (PMIC) | The PM8916 power management IC provides voltage regulation for the processor, flash/RAM and other parts; it also provides audio out to the speaker and controls the LCD backlight. |

***Table 5:** The head-boards functional elements*

| | |
|---|---|
| speaker | A speaker is used to play sounds, and for speech synthesis |
| WiFi transceiver | An 802.11AC WiFi transceiver is built into the processor package |

## 10.1. THE APQ8009 PROCESSOR

The head-board is based on the Qualcomm “Snapdragon 212” APQ8009 SOC. It is a quad-core processor; each core is a 32-bit ARM Cortex A7. It also includes a DSP (“Hexagon 536”), and GPU (Adreno 304). It also includes WiFi and Bluetooth LE transceivers. The processor has interfaces to external memory, for the camera (using MIPI), the display, and the audio playback.

The APQ8009 processor is a sibling to the MSM8909 processor employed in cell phones, where APQ is short for “application processor Qualcomm” and MSM is short for “mobile station modem.” The difference is that the later includes some form of modem, such as HPSA, CDMA, or LTE. Both designators are used in software code-bases employed with Vector. The most likely reason is the naming of registers, drivers, and other useful software didn't carefully limit the use of MSMxxxx references to just the processors with modems.

The flash & RAM are connected to the processor on SDHC1. The device tree file shows that during development Vector’s also supported an SD card slot on SDHC2.

## 10.2. SPEAKERS

The speaker is driven at 16bits, single channel, with a sample rate of 8000-16025 samples/sec.

## 10.3. CAMERA

Vector has a 720p camera with a 120° field of view. The camera is calibrated at manufacturing time. The camera vertical sync (frame sync) is connected to the interrupt input on the IMU to synchronize the samples.

***Table 6:*** *The camera controls*

| GPIO | Description |
|---|---|
| 26 | Camera interface clock |
| 48 | Camera reset |
| 83 | Camera power enable (from PM8916 PMIC) |
| 94 | Camera standby |

## 10.4. THE LCD

*LCD display*

Vector’s LCD is a backlit IPS display assembly made by Truly. The processor is connected to the LCD via SPI. Two LEDs are used to illuminate the LCD. The backlight is PWM controlled by the PM8916 PMIC.

US Patent 20372659

The prior generation, Cozmo, used an OLED display for his face and eyes. OLEDs are susceptible to burn-in and uneven dimming or discoloration of overused pixels. Anki addressed this with two accommodations. First it gave the eyes regular motion, looking around and blinking. Second, the illuminated rows were regularly alternated to give a retro-technology interlaced row effect, like old CRTs.

Vector's IPS display gives a smoother imagery that is much less susceptible to burn-in, at the expense of higher power.

## 10.5. TRIM, CALIBRATION SERIAL NUMBERS AND KEYS

Each Vector has a set of per unit calibrations:

- The camera is calibrated
- The IMU is calibrated
- The motor power is calibrated[15]

There are per unit keys, MAC addresses and serial numbers

- Each processor has its own unique key, used to with the Trust Zone
- The WiFi and Bluetooth have assigned, unique MAC addresses.
- Each Vector has an assigned serial number

## 10.6. MANUFACTURING TEST CONNECTOR/INTERFACE

It is a common practice to include at least one interface on a product for use during manufacture. This is used to load software and firmware, unique ids – WiFi MACs, serial number – to perform any calibration steps and to perform run-up checks that the device functions / is assembled correctly. It is intended to be a fast interface that doesn't cause yield fallout. Typically (but there are exception) this is not radio based, as they can interfere or have fiddly issues.

The USB interface is used to load firmware. The microprocessors include a built-in boot-loader (ABOOT), which includes support for loading firmware into the devices flash.

For the other functions, there are three possibilities

- There is a UART, that provides a boot console, but does not accept input
- There is a USB connector that probably is used to load firmware.
- The WiFi, once MAC addresses have been loaded into the unit

# 11. REFERENCES & RESOURCES


Amitabha, *Benchmarking the battery voltage drain in Anki Vector and Cozmo*, 2018 Dec 31
https://medium.com/programming-robots/benchmarking-the-battery-voltage-drain-in-anki-vector-and-cozmo-239f23871bf8

Anki, *Lithium single-cell battery data sheet*
https://support.anki.com/hc/article_attachments/360018003653/Material%20Safety%20Data%20Sheet_April%202018.pdf

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

Diodes Inc, *DMG2305UX P-Channel Enhancement Mode MOSFET*
https://www.diodes.com/assets/Datasheets/DMG2305UX.pdf

Diodes, Inc, *DMC2038LVT Complementary Pair Enhancement Mode MOSFET*
https://www.diodes.com/assets/Datasheets/products_inactive_data/DMC2038LVT.pdf


[15] Todo: look into what this means. R43 looks like a sense resistor.. is this a current sense for power? Does Vector also use it as a poor mans (discharge only) Coulomb counter?

Everlight *EAAPMST3923A2*

Kingston Technology, *Embedded Multi-Chip Package 04EMCP04-NL3DM627-Z02U*, rev 1.2, 2016
https://cdn.discordapp.com/attachments/573889163070537750/595223765206433792/04EMCP04-NL3DM627-Z02U_-_v1.2.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

Panda, a data sheet for a similar single-cell lithium battery
https://panda-bg.com/datasheet/2408-363215-Battery-Cell-37V-320-mAh-Li-Po-303040.pdf

Qualcomm, APQ8009 Processor
https://www.qualcomm.com/products/apq8009

Qualcomm, *PM8916/PM8916-2 Power Management ICs Device Specification*, Rev C, 2018 Mar 13
https://developer.qualcomm.com/qfile/29367/lm80-p0436-35_c_pm8916pm8916_power_management_ics.pdf

ST Microelectronics, *STM32F030x8*, Rev 4, 2019-Jan
https://www.st.com/en/imaging-and-photonics-solutions/vl53l1x.html

ST Microelectronics.  Touch sensing
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

ST Microelectronics.  VL531L1 Long distance ranging Time-of-Flight sensor
https://www.st.com/content/ccc/resource/technical/document/application_note/group0/ed/0d/4
https://www.st.com/resource/en/datasheet/vl53l1x.pdf

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

CHAPTER 3

# Accessory Electronics design description

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

- The charging station
- The companion cube

## 12. CHARGING STATION

The charging station is intended to provide energy to the Vector, allowing it to recharge.

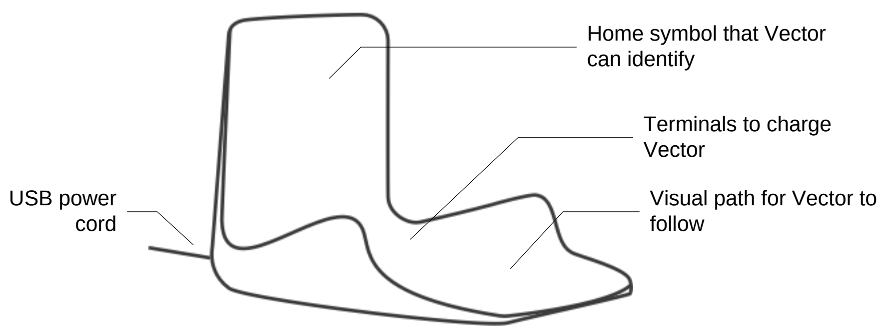



***Figure 13:*** *Charging station main features*

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 supply +5V (from the power adapter) to Vector, allowing him to recharge. The terminals are offset in such a way to prevent Vector from accidentally being subject to the wrong polarity. Vector has to be backed into charging station in mate with the connectors. Vector face-first, even with his arms lifted, will not contact the terminals.

The charging station has an optical marker used by Vector to identify the charging station and its pose (see chapter 15).

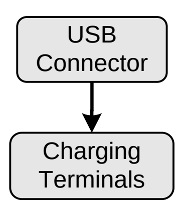



***Figure 14:*** *Charging station block diagram*

## 13. 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 15).

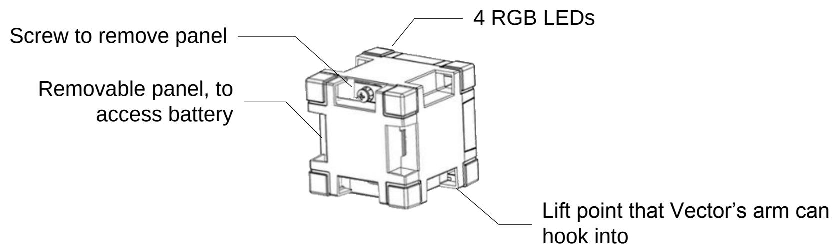



***Figure 15:** Cube's main features*

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:

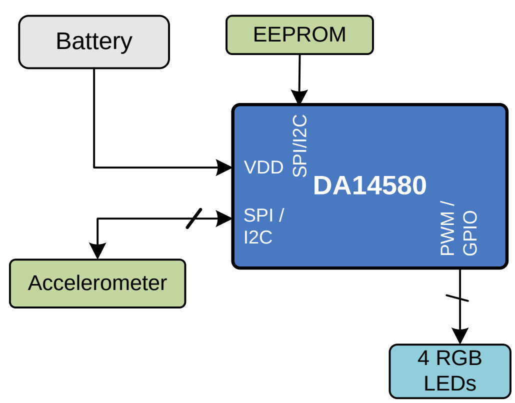



***Figure 16:** Block diagram of the Cube's electronics*

The Cube's electronic design includes the following elements:

***Table 7:** The Cube's electronic design 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.[16] |
| 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 |

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

| | |
|---|---|
| | have independent colors. |

The communication protocol is given in Appendix F.

## 13.1. OVER THE AIR FIELD UPDATES

The DA14580 has a minimal ROM boot loader that initializes hardware, moves a secondary boot loader 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 boot loader loads the application firmware from I2C or SPI EEPROM or flash to SRAM and pass control to it.

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

## 13.2. REFERENCES & 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 Vector’s software design.

- THE SOFTWARE ARCHITECTURE. A detailed look at Vector’s overall software architecture and main modules.
- STARTUP. A detailed look at Vector’s startup and shutdown processes
- POWER MANAGEMENT. A detailed look at Vector’s architecture for battery monitoring, changing and other power management.
- BASIC INPUT AND OUTPUT. A push button, touch sensing, surface proximity sensors, time of flight proximity sensing, and backpack LEDs.
- AUDIO INPUT AND OUTPUT.

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

# CHAPTER 4

# Architecture

This chapter describes Vector's software architecture:

- The architecture
- The emotion-behaviour system
- The communication infrastructure
- Internal support

## 14. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE

Vector's architecture has a structure something like:

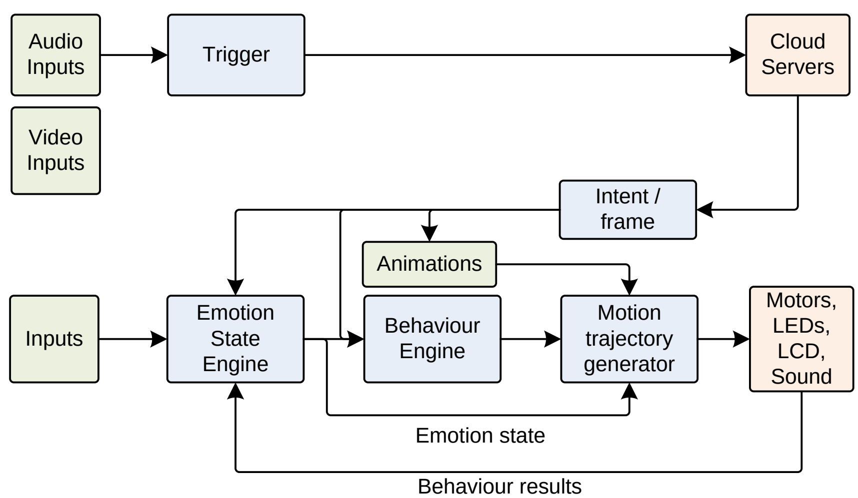



***Figure 17:** The overall functional block diagram*

Fast control loops — to respond quickly — are done on the Vector's hardware. Other items, processing heavy including (but not limited to) speech recognition, natural language processing, and training for faces are sent to the cloud.

Vector is built on a version of Yocto Linux. Anki selected this for a balance of reasons: some form of Linux is required to use the Qualcomm processor, the low up front (and royalty) costs, the availability of tools and software modules. The Qualcomm is a multi-processor, with four main processing cores and a GPU. Vector runs a handful of different application programs, in addition to the OS's foundational service tasks and processes.

*explored in* Casner, *and* Wiltz

## 14.1. APPLICATION SERVICES ARCHITECTURE

The application is divided into the following services

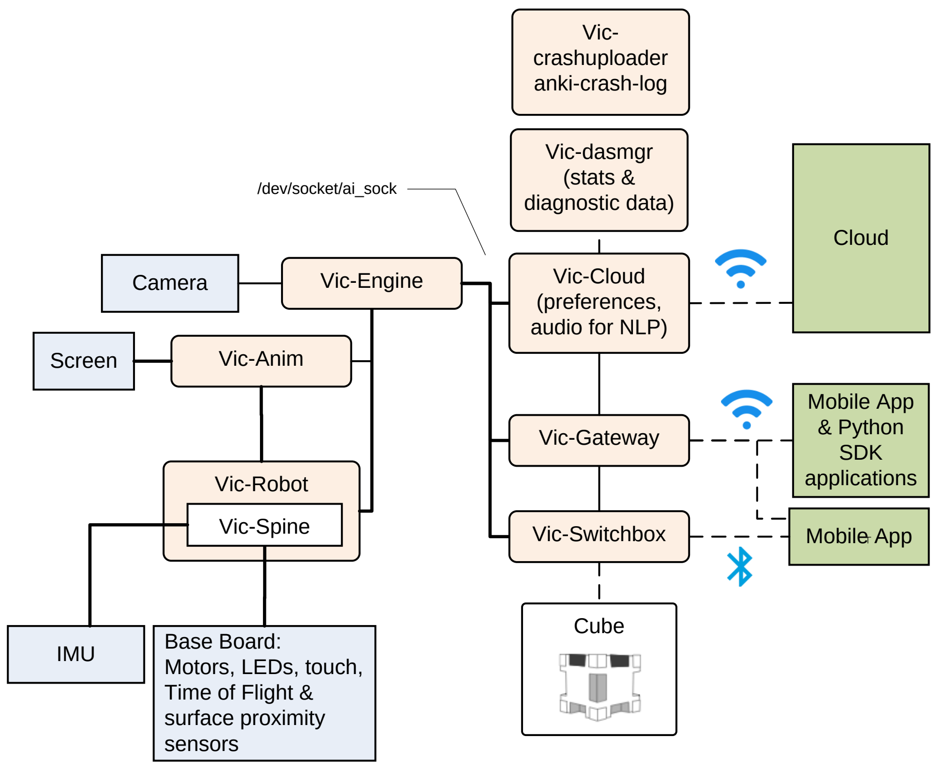



***Figure 18:*** *The overall communication infrastructure*

There are multiple applications that run:

***Table 8:*** *Vector processes*

| Services | Speculated purpose |
|---|---|
| vic-anim | Probably plays multi-track animations (which include motions as well as LCD display and sound)<br>config file: /anki/etc/config/platform_config.json /anki/data/assets/cozmo_resources/webserver/webServerConfig_anim.json |
| vic-bootAnim | LCD and sound animations during boot |
| vic-cloud | Probably connects to the cloud services for natural language |
| vic-crashuploader anki-crash-log | A service that sends logs (especially crash logs and mini-dumps) to remote servers for analysis. |
| vic-dasmgr | Gathering data on processor and feature usage, possibly intended to serve as a foundation for gathering data when performing experiments on settings and features. |
| vic-engine | The behaviour / emotion engine. Hooks into the camera face recognizer. |
| vic-gateway | Responsible for the local API/SDK services available as gRPC services on https. |
| vic-robot | Basic power management. Resets watchdog timer. Internally has "vic-spine" that manages the sensors. |
| vic-switchboard | Supports the Bluetooth LE communication interface, including the mobile application protocol (see chapter 12). Routes messages between the other |

| | |
|---|---|
| | services? Manages the access keys |

## 14.2. EMOTION MODEL, BEHAVIOUR ENGINE, ACTIONS AND ANIMATION ENGINE

Vector's high-level AI is organized around an *emotion model*, and a *behaviour engine* that drives actions. "Behaviors represent a complex task which requires Vector's internal logic to determine how long it will take. This may include combinations of animation, path planning or other functionality." Behaviours have states, which can initiate actions and can accept different intents in the different states.

Anki Vector SDK

Animations are scripted, highly-coordinated motions, and sounds (as well as displayed items) that Vector carries out. Behaviours and intents (response to voice interaction) can initiate animations. However, they do not initiate a specific animation script. Instead the specify a kind of animation – referred to by an *animation trigger name* – and the animation engine selects the specific animation based on context and current emotional state.

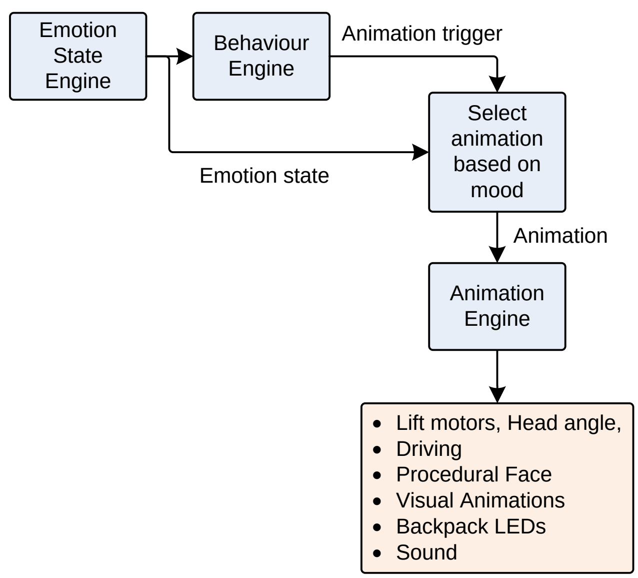



***Figure 19:*** *The behaviour-animation flow*

# 15. STORAGE SYSTEM

Vector's system divides the storage into many regions, primarily based on whether the region is modifiable (and when), and which subsystem manages the data. Appendix E describes the flash partitions and file system structure. See chapter 6 for a description of the partitions used for system start up and restore.

Most of the partitions on the flash storage are not modifiable – and are checked for authenticity (and alteration). These partitions hold the software and assets as delivered by Anki (and Qualcomm) for a particular release of the firmware. They are integrity checked as part of the start procedure. (See Chapter 6 for a description.)

Data that is specific to the robot, such as settings, security information, logs, and user data (such as pictures) are stored in modifiable partitions. Some of this data is erased when the unit is "reset" to factory conditions

These are described below.

## 15.1. ELECTRONIC MEDICAL RECORDS (EMR)

Vector's "Electronic Medical Record" (EMR) partition holds the following information:

| Offset | Size | Type | Field | Description |
|---|---|---|---|---|
| 0 | 4 | uint32_t | *ESN* | Vector's electronic serial number (ESN). This is the same serial number as printed on the bottom of Vector. |
| 4 | 4 | uint32_t | *HW_VER* | Hardware revision code |
| 8 | 4 | uint32_t | *MODEL* | The model number of the product |
| 12 | 4 | uint32_t | *LOT_CODE* | The manufacturing lot code |
| 16 | 4 | uint32_t | *PLAYPEN_READY_FLAG* | The manufacturing fixture tests have passed, it ok to run play pen tests. |
| 20 | 4 | uint32_t | *PLAYPEN_PASSED_FLAG* | Whether or not Vector has passed the play pen tests. |
| 24 | 4 | uint32_t | *PACKED_OUT_FLAG* | |
| 28 | 4 | uint32_t | *PACKED_OUT_DATE* | (In unix time?) |
| 32 | 192 | uint32_t[4] | *reserved* | |
| 224 | 32 | uint32_t[8] | *playpen* | |
| 256 | 768 | uint32_t[192] | *fixture* | |

***Table 9:*** *Electronic Medical Record (EMR)*

This information is not modified after manufacture; it persists after a device reset or wipe.

## 15.2. OEM PARTITION FOR OWNER CERTIFICATES AND LOGS

The OEM partition is a read/writeable ext2 filesystem. It is used to hold the SDK certificate folders:

| Folder | Description |
|---|---|
| | The top level holds the log files. |
| *cloud* | Holds the SDK TLS certificate and signing keys. With newer firmware, the folder may also hold some other calibration information. |
| *nvStorage* | holds some binary "`.nvdata`" files |

***Table 10:*** *OEM partition file hierarchy*

# 16. SECURITY AND PRIVACY

*Anki Security & Privacy Policy*

Vector's design includes a well thought-out system to protect against disclosing (i.e. providing to strangers) sensitive information, and allowing the operator to review and delete it at any time:

- Photographs taken by Vector are not sent to (nor stored in) a remote server. They are stored in encrypted file system, and only provided to authenticated applications on the local network. Each photograph can be individually deleted (via the mobile application).
- The image stream from Vector's camera is not sent to a remote server. It is only provided to authenticated applications on the local network.

- The data used to recognize faces[17] and the names that Vector knows are not sent to (nor stored in) a remote server. The information is stored in encrypted file system. The list of known faces (and their names) is only provided to authenticated applications on the local network. Any facial recognition data not associated with a name is deleted when Vector goes to sleep. Facial data associated with an individual name can be deleted (along with the name) via the mobile application.
- "[After] you say the wake words, "Hey Vector", Vector streams your voice command to the cloud, where it is processed. Voice command audio is deleted after processing. Text translations of commands are saved for product improvement not associated with a user."
- The audio stream from the microphone — if it had been finished being implemented – would have been provided to authenticated applications on the local network.
- Information about the owner
- Control of the robots movement, speech & sound, display, etc. is limited to authenticated applications on the local network.

Vector's software is protected from being altered in a way that would impair its ability to secure the above.

Vector also indicates when it is doing something sensitive:

- When the microphone is actively listening, it is always indicated on the backpack lights (blue).
- The microphone is enabled by default, but only listening for the wake word, unless Vector's microphone has been disabled.
- When the camera is taking a picture (to be saved), Vector makes a sound
- When the camera is XYZ on?
- Unless the backpack lights are all orange, the WiFi is enabled. (All orange indicates it is disabled.)

## 16.1. ENCRYPTED COMMUNICATION

The file system with personally identifying information and other data about the owner — photos, account information, WiFi passwords, and so one — is encrypted.

## 16.2. ENCRYPTED FILESYSTEM

The file system with the users data — photos, account information, WiFi passwords, and so one — is encrypted.

## 16.3. THE OPERATING SYSTEM

There is a chain of firmware signed by Anki. This is intended to protect Vector's software from being altered in a way that would impair its ability to secure the above information.

Android boot loaders typically include a few powerful (but unchecked) bits that disable the signature checking, and other security features. These bits typically are set either thru commands

[17] The Anki privacy and security documents logically imply that the face image is not sent to Anki servers to construct a recognition pattern. I would have guessed heavy crunching would have been performed on a server.

to the firmware during boot up, by applications, or possibly by hack/exploit. Sometimes this requires disassembling the device and shorting some pins on the circuit board.

Vector doesn't support those bits, nor those commands. Signature checking of the boot loader, kernel and RAM disk can't be turned off.

### 16.3.1 The possibility for future modifications to Vector's software

There may be a way to disable checking of the system file system and its software.

Anki created special Vectors for internal development. The software for these units has a special version of the kernel and RAM disk that does not check system room file system, and makes it writable. This file system has Vector's application soft, supports SSH. This software was tightly controlled, and "only .,. available inside the Anki corporate network." For purposes of customizing and updating Vector, this version is essential. (Note: the kernel and RAM disk can't be modified.) Jane Fraser, 2019

Note: the OTA software has a "dev" (or development) set of OTA packages. Those packages are not the same; they are essential software release candidates being pushed out for test purposes.

## 16.4. AUTHENTICATION

The web services built into Vector require a token. This is used to prove that you have authenticated (with the more capable — and not physically accessible — servers). This authentication is to protect:

- Photos already on Vector
- The image stream from the camera
- The audio stream from the microphone — if it had finished being implemented
- The sensitive owner information
- Controlling the robot

(That is to say, to prevent disclosure)

Once authenticated you can access those, as

# 17. CONFIGURATION AND ASSET FILES

The Anki vector software is configured by JSON files. Some of the JSON files were probably created by a person (for the trivial ones). Others were created by scripting / development tools; a few of these were edited by developers. These JSON files are clearly intended to be edited by people:

- The files are cleanly spaced, not in the most compact minimized size
- The JSON parser supports comments, which is not valid JSON. Many files have comments in them. Many have sections of the configuration that are commented out.

## 17.1. CONFIGURATION FILES

the .env files are KEY=VALUE, used by the systems start up of the devices

and some of the scripts

The top-level configuration file provides the paths to the network other configuration files. It is found at:

/anki/etc/config/platform_config.json

This path is hardcoded into the vic-dasmgr, and provided in the editable startup files for vic-anim and vic-engine. The configuration file contains a JSON structure with the following fields:

| Field | Value | Description & Notes |
| --- | --- | --- |
| *DataPlatformCachePath* | "/data/data/com.anki.victor/cache" | |
| *DataPlatformPersistentPath* | "/data/data/com.anki.victor/persistent" | |
| *DataPlatformResourcesPath* | "/anki/data/assets/cozmo_resources" | The path to most configuration files and assets |

***Table 11:*** *The platform config JSON structure*

When describing the configuration and asset files, a full path will be provided. When the path is constructed from different parts, the part that is specified in another configuration or binary file will be outlined. The path to a settings file might look like:

/anki/assets/cozmo_resources/config/engine/settings_config.json

The path leading up to the settings file (not outlined in red) is specified in an earlier configuration file, usually the platform configuration file described above.

## 18. SOFTWARE-HARDWARE LAYERS

- Base-board input/output software architecture
- The LCD display
- Camera

### 18.1. THE BASE BOARD INPUT/OUTPUT

The base-board input-output software has a structure like so:

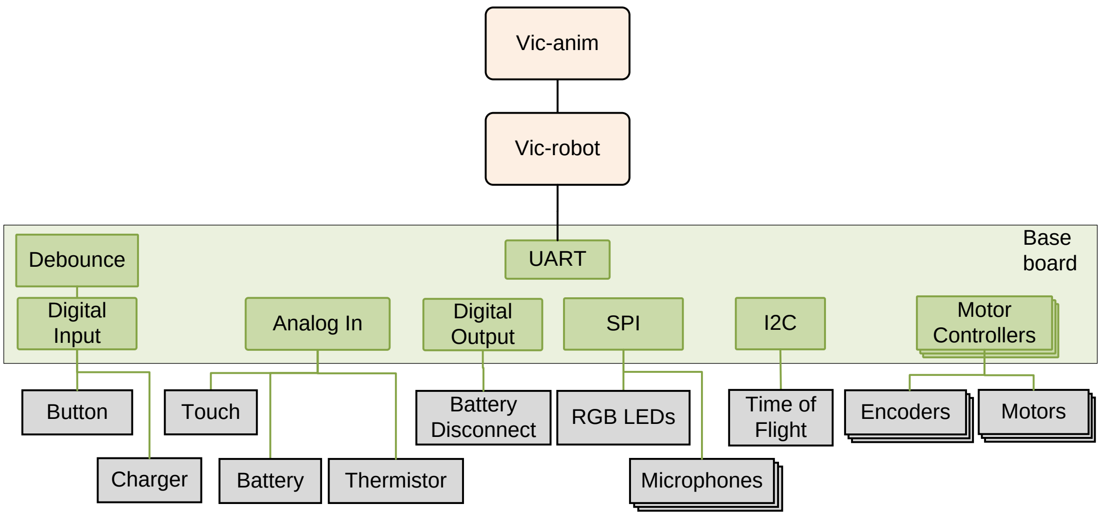



***Figure 20:*** *The baseboard-related architecture*

### 18.2. THE LCD DISPLAY

Four different applications may access the display, albeit not at the same time:

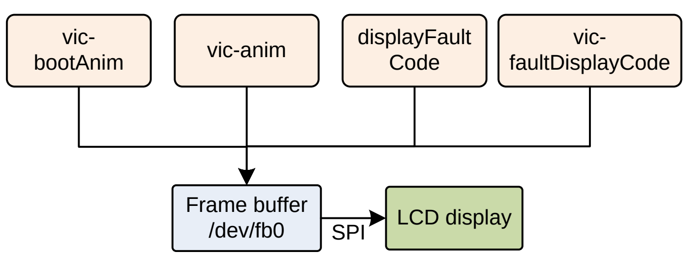



***Figure 21:*** *The LCD architecture*

Note: `displayFaultCode` is present on Vector, but it is not called by any program.

The LCD is connected to the MPU via an SPI interface (`/dev/spidev1.0`). The frame buffer (`/dev/fb0`) is essentially a buffer with metadata about its width, height, pixel format, and orientation. Application modifies the frame buffer by `write()` or `memmap()` and modifies the bytes. Then the frame buffer has the bytes transfer (via SPI) tot the display.

`vic-anim` employs a clever screen compositing system to create Vector's face (his eyes), animate text jumping and exploding, and small videos, such as rain or fireworks.

The `vic-faultDisplayCode` and Customer Care Information Screen of `vic-anim` have a visual aesthetic is unlike the rest of Vector. These modes employ a barebones system for the display.

The text appears to rendered into the buffer using OpenCV's `putText()` procedure, and transferring it to the display without any further compositing.

Not sure if the transfer is in a driver, in the kernel, or in user space... or which process would have it in user space.

## 18.3. THE CAMERA

The camera subsystem has the following architecture:

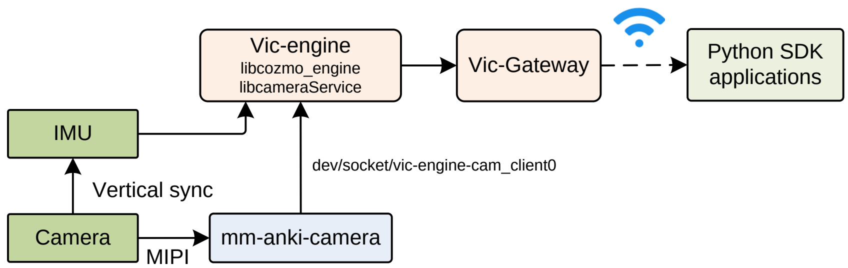



***Figure 22:*** *The camera architecture*

The camera's vertical synchronization signal is connect to the interrupt line on IMU, triggering accelerometer and gyroscope sampling in sync with the camera frame. The vision is used as a navigation aid, along with the IMU data. The two sources of information are fused together in the navigation system to form a more accurate position and relative movement measure. The image must be closely matched in time with the IMU samples. However the transfer of the image from the camera to the processor, then thru several services to vic-engine introduces variable or unpredictable delays. The camera's vertical sync – an indication of when the image is started being sampled – is used to trigger the IMU to take a sample at the same time.

Daniel Casner, 2019 Embedded Vision Summit

The camera is also used as an ambient light sensor when Vector is in low power mode (e.g. napping, or sleeping). In low power mode, the camera is suspended and not acquiring images. Although in a low power state, it is still powered. The software reads the camera's auto exposure/gain settings and uses these as an ambient light sensor. (This allows it to detect when there is activity and Vector should wake.)

# 19. REFERENCES & RESOURCES

Anki, *Elemental Platform*
https://anki.com/en-us/company/elemental-platform.html

Describes, as a marketing brochure, much of Anki's product network architecture.

Anki, *Vector Security & Privacy FAQs*, 2018
https://support.anki.com/hc/en-us/articles/360007560234-Vector-Security-Privacy-FAQs

Casner, Daniel, *Consumer Robots from Smartphone SoCs*, Embedded Systems Conference Boston, 2019 May 15
https://schedule.esc-boston.com//session/consumer-robots-from-smartphone-socs/865645

Stein, Andrew; Kevin Yoon, Richard Alison Chaussee, Bradford Neuman, Kevin M.Karol, US Patent US2019/01563A1, *Custom Motion Trajectories for Robot Animation*, Anki, filed 2018 Jul 13, published 2019 Apr 18,

Qualcomm, *Snapdragon™ 410E (APQ8016E) r1034.2.1 Linux Embedded Software Release Notes*, LM80-P0337-5, Rev. C, 2018 Apr 10
lm80-p0337-5_c_snapdragon_410e_apq8016e_r1034.2.1_linux_embedded_software_revc.pdf

Wiltz, Chris, *Lessons After the Failure of Anki Robotics*, Design News, 2019 May 21 https://www.designnews.com/electronics-test/lessons-after-failure-anki-robotics/140103493460822

# CHAPTER 5

# Startup

This chapter describes Vector's start up and shutdown processes:

- The startup process
- The shutdown steps

## 20. STARTUP

Vector's startup is based on the Android boot loader and linux startup.[18] These are otherwise not relevant to Vector, and their documentation is referred to. The boot process gets quite far before knowing why it booted up or being able to response in complex fashion.

1. The backpack button is pressed, or Vector is placed into the charger. This powers the base board, and the head-boards.
2. The base-board displays an animation of the backpack LEDs while turning on. If turned on from a button press and the button is released before the LED segments are fully lit, the power will go off. If the held long enough, the head-board will direct the base-board to keep the battery switch closed.

### 20.1. QUALCOMM'S PRIMARY AND SECONDARY BOOTLOADER

Meanwhile, on the head-board:

1. "Qualcomm's Primary Boot Loader is verified and loaded into [RAM] memory[19] from BootROM, a non-writable storage on the SoC. [The primary boot loader] is then executed and brings up a nominal amount of hardware," Nolen Johnson
2. The primary boot loader checks to see if a test point is shorted on the board, the unit will go into emergency download (EDL) mode. It is known the when F_USB pad on the head-board is pulled to Vcc, USB is enabled; this may be the relevant pin. Roee Hay
3. If the primary boot loader is not in EDL mode it "then verifies the signature of the next boot loader in the chain [the secondary bootloader], loads it, [and] then executes it." The secondary boot loader is stored in the flash partition SBL. Nolen Johnson
4. If the secondary boot loader does not pass checks, the primary boot loader will go into emergency down load mode.
5. "The next boot loader(s) in the chain are SBL*/XBL (Qualcomm's Secondary/eXtensible Boot Loader). These early boot loaders bring up core hardware like CPU cores, the MMU, etc. They are also responsible for bringing up core processes .. [for] TrustZone. The last

[18] An ideal embedded system has a fast (seemingly instant) turn on. Vector's startup *isn't* fast. The steps to check the data integrity of the large flash storage – including checking the security signatures – and the complex processes that linux provides each contribute to the noticeable slow turn on time. Checking the signatures is inherently slow, by design.
[19] The boot loader is placed into RAM for execution to defeat emulators.

purpose of SBL*/XBL is to verify the signature of, load, and execute aboot/ABL [Android boot loader]."
The Android boot loader (aboot) is stored on the "ABOOT" partition.
The secondary bootloader also supports the Sahara protocol; it is not known how to activate it.

Note: The keys for the boot loaders and TrustZone are generated by Qualcomm, with the public keys programmed into the hardware fuses before delivery to Anki or other customers. The signed key pair for the secondary boot loader is not necessarily the same signed key pair for the aboot. They are unique for each of Qualcomm's customer. Being fuses they cannot be modified, even with physical access.

## 20.2. ANDROID BOOTLOADER (ABOOT)

1. "Aboot brings up most remaining core hardware then in turn normally verifies the signature of the boot image, reports the verity status to Android Verified boot through dm-verity… On many devices, Aboot/ABL can be configured to skip cryptographic signature checks and allow booting any kernel/boot image."

    a. On other devices, aboot reads the DEVINFO partition for a structure. It checks the header of the structure for a magic string ("ANDROID-BOOT!") and then uses the values internally to indicate whether or not the device is unlocked, whether verity-mode enabled or disabled, as well as a few other settings. By writing a version of this structure to the partition, the device can be placed into unlock mode. Roee Hay

       Vector does *not* support this method of unlocking.

    b. "The build system calculates the SHA256 hash of the raw boot.img and signs the hash with the user's private key… It then concatenates this signed hash value at the end of raw boot.img to generate signed boot.img." Qualcomm LM80 P0436

    c. "During bootup, [Aboot[20]] strips out the raw boot.img and signed hash attached at the end of the image. [Aboot] calculates the SHA256 hash of the complete raw boot.img and compares it with the hash provided in the boot.img. If both hashes match, kernel image is verified successfully."

2. ABoot can either program the flash with software via boot loader mode, or load a kernel. The kernel can be flagged to use a recovery RAM disk or mount a regular system.

3. If recovery mode, it will load the kernel and file systems from the active RECOVERY partitions.

    a. Recovery is entered if the active regular partition cannot be loaded, e.g. doesn't exist or fails signature check, or

    b. The RX signal from the base-board is held low when aboot starts.[21] If this is the case, "anki.unbrick=1" is prepended to the command line passed to the kernel.

4. ABoot loads the kernel and RAM file system from the active "BOOT" partition and passes it command line to perform a check of the boot and RAM file system the signatures.[22] The

[20] The Qualcomm document speaks directly about Little Kernel; ABoot is based on Little Kernel.
[21] This seems risky. It implies that (a) the baseboard doesn't communicate until the head board sends something to it; and (b) a reset/start that cause the bootloader to run again is either very rare or protected against by a reset-reason flag.
[22] The check specifies the blocks on the storage to perform a SHA256 check over and provides expected signature result.

command line is stored in the header of the boot partition; it is checked as part of the signature check of the boot partition and RAM file system.

Many of these elements will be revisited in Chapter 20 where updating aboot, boot, and system partitions are discussed.

## 20.3. REGULAR SYSTEM BOOT

The boot partition kernel and RAM disk begin an Anki-specific system check:

1. The RAM file system contains two programs: init and /bin/rampost. init is a shell script and the first program launched by the kernel. This script calls rampost to turn on the LCD, its backlight and initiate communication with the base-board. (This occurs ~6.7 seconds after power-on is initiated).
    a. rampost will perform a firmware upgrade of the base-board if its version is out of date. It loads the firmware from syscon.dfu (Note: the firmware in the base-board is referred to as syscon.)
    b. rampost checks the battery voltage, temperature and error flags. It posts any issues to /dev/rampost_error
    c. All messages from rampost are prefixed with "@rampost."
2. Next, init performs a signature check of the system partition to ensure integrity. This is triggered by the command line which includes dm-verity options prefixed with "dm=". If the system does not pass checks, init fails and exits.[23]
    a. Note: none of the file systems in fstab marked for verity checking, so this is the only place where it is performed.
3. The main system file-system is mounted and launches the main system initialization.

The regular boot uses systemd to allow of the startup steps to be performed in parallel. The rough start up sequence is:

1. Starts the Qualcomm Secure Execution Environment Communicator (dev-qseecom.device) and ION memory allocator (dev-ion.device)
2. The encrypted user file system is checked and mounted (via the mount-data service). This file system is where the all of the logs, people's faces, and other information specific to the individual Vector are stored. The keys to this file system are stored in the TrustZone in the MPU's SOC fuse area. This file system can only be read by the MPU that created it.
3. The MPU's clock rate is limited to 533Mhz, and the RAM is limited to 400MHz to prevent overheating.
4. The camera power is enabled
5. If Vector doesn't have a robot name, Vic-christen is called to give it one.
6. After that several mid-layer communication stacks are started:
    a. usb-service any time after that
    b. the WiFi connection manager (connman)

[23] TBD what happens for recovery?

c. The time client (chronyd), to retrieve network time. (Vector does not have a clock that keeps time when turned off)

d. init-debuggerd

7. multi-user, sound, init_post_boot
8. The "Victor Boot Animator" is start (~8 seconds after power on). This is probably the sparks.
9. Victor Boot completes ~20.5 after power on, and the post boot services launches
10. vic-crashloader
11. vic-robot
12. Once the startup has sufficiently brought up enough the next set of animations the sound of boot
13. VicOS is running ~32 seconds after power on. The boot is complete, and Vector is ready to play

## 20.4. ABNORMAL SYSTEM BOOT

If there is a problem during startup – such as one of the services is unable to successfully start, a fault code associated with that service is stored in `/run/fault code` and the fault code displayed. See chapter 21 for a description of the display of fault codes and diagnostics. See Appendix C for fault codes.

# 21. SHUTDOWN

- Turning Vector off manually
- Vector turning off spontaneously due to brown-out or significant loss of power
- Vector turning off (under low power) by direction of the head-board

Vector cannot be turned off via Bluetooth LE, or the local gRPC SDK access. There are no exposed commands that do this. Using a verbal command, like "turn off" does not direct Vector to shut off (disconnect the battery). Instead it goes into a quiet mode. Although it may be possible for a Cloud command to turn Vector off, this seems unlikely.

However, there is likely a command to automate the manufacture and preparation for ship process.

## 21.1. TURNING VECTOR OFF (INTENTIONALLY)

There is a shutdown code that tracks the reason for shutdown:

***Table 12:** Vector shutdown codes*

| Element | Description & Notes |
|---|---|
| SHUTDOWN_BATTERY_CRITICAL_TEMP | Vector shut down automatically because the battery temperature was too high. |
| SHUTDOWN_BATTERY_CRITICAL_VOLT | Vector shut down automatically because the battery voltage was too low. |
| SHUTDOWN_BUTTON | Vector was shut down by a long button press. |
| SHUTDOWN_GYRO_NOT_CALIBRATING | Vector shut down automatically because of an IMU problem(?) |

| | |
|---|---|
| SHUTDOWN_UNKNOWN | Vector shut down unexpectedly; the reason is not known. Likely a brown-out or battery voltage dipped low faster than Vector could respond to. |

It is not clear where the shutdown code is stored

### 21.2. UNINTENTIONALLY

The base-board is responsible for keeping the battery connected. However brownouts, self-protects when the voltage get to too low, and bugs can cause the battery to be disconnected.

### 21.3. GOING INTO AN OFF STATE

When Vector wants to intentionally turn off, it cleans up its state, to gracefully shutdown the linux system and tells the base-board to disconnect the battery.

## 22. REFERENCES & RESOURCES


Android, *Verified Boot*
https://source.android.com/security/verifiedboot/

Bhat, Akshay; *Secure boot on Snapdragon 410*¸TimeSys, 2018 Aug 23
https://www.timesys.com/security/secure-boot-snapdragon-410/

Discusses how one can get the source to the secondary bootloader (SBL), the tools to sign it and aboot using sectools

https://gitlab.com/cryptsetup/cryptsetup/wikis/DMVerity

Hay, Roee. *fastboot oem vuln: Android Bootloader Vulnerabilities in Vendor Customizations*, Aleph Research, HCL Technologies, 2017
https://www.usenix.org/system/files/conference/woot17/woot17-paper-hay.pdf

Hay, Roee; Noam Hadad. *Exploiting Qualcomm EDL Programmers*, 2018 Jan 22
*Part 1: Gaining Access & PBL Internals*
https://alephsecurity.com/2018/01/22/qualcomm-edl-1/
*Part 2: Storage-based Attacks & Rooting*
https://alephsecurity.com/2018/01/22/qualcomm-edl-2/
*Part 3: Memory-based Attacks & PBL Extraction*
https://alephsecurity.com/2018/01/22/qualcomm-edl-3/
*Part 4: Runtime Debugger*
https://alephsecurity.com/2018/01/22/qualcomm-edl-4/
*Part 5: Breaking Nokia 6's Secure Boot*
https://alephsecurity.com/2018/01/22/qualcomm-edl-5/

Johnson, Nolen; *Qualcomm's Chain of Trust*, Lineage OS, 2018 Sept 17
https://lineageos.org/engineering/Qualcomm-Firmware/

A good overview of Qualcomm's boot loader, boot process, and differences between versions of Qualcomm's process. Quotes are slightly edited for grammar.

Nakamoto, Ryan; *Secure Boot and Image Authentication*, Qualcomm , 2016 Oct
https://www.qualcomm.com/media/documents/files/secure-boot-and-image-authentication-technical-overview-v1-0.pdf

Qualcomm, *DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor Little Kernel Boot Loader Overview*, LM80-P0436-1, Rev D, 2016 Jul
lm80-p0436-1_little_kernel_boot_loader_overview.pdf

https://github.com/ alephsecurity
A set repositories researching tools to discover commands in Sahara and EDL protocols

https://github.com/openpst
A set of repositories researching and implementing an interface to the Sahara protocol.

## CHAPTER 6

# Power management

This chapter describes Vector's power management:

- The battery management
- Load shedding
- Charger info?

## 23. POWER MANAGEMENT

### 23.1. BATTERY MANAGEMENT

There isn't a coulomb counter to track the remaining energy in the battery. At the broadest strokes, the battery voltage is used to predict the battery state of charge.

#### 23.1.1 Battery levels

Vector maps the battery voltage into a battery level, taking into account whether or not the charger is active:

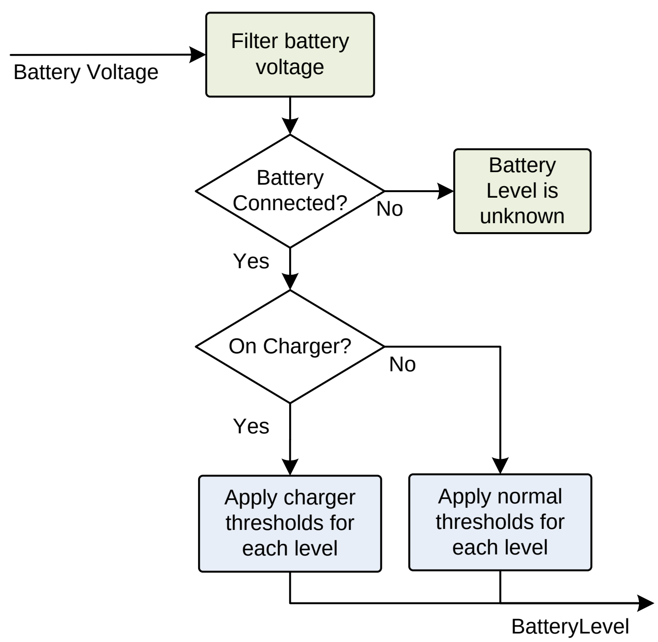



***Figure 23:*** *The battery level classification tree*

Note: The battery voltage is filtered as the voltage will bounce around with activity by the motors, driving the speaker and processors.

The `BatteryLevel` enumeration is used to categorize the condition of the Vector battery:

***Table 13:** BatteryLevel codes[24] as they apply to Vector*

| Name | Value | Description |
|---|---|---|
| *BATTERY_LEVEL_FULL* | 3 | Vector's battery is at least 4.1V |
| *BATTERY_LEVEL_LOW* | 1 | Vector's battery is 3.6V or less; or if Vector is on the charger, the battery voltage is 4V or less. |
| *BATTERY_LEVEL_NOMINAL* | 2 | Vector's battery level is between low and full. |
| *BATTERY_LEVEL_UNKNOWN* | 0 | If the battery is not connected, Vector can't measure its battery. |

The battery levels are organized conventionally:

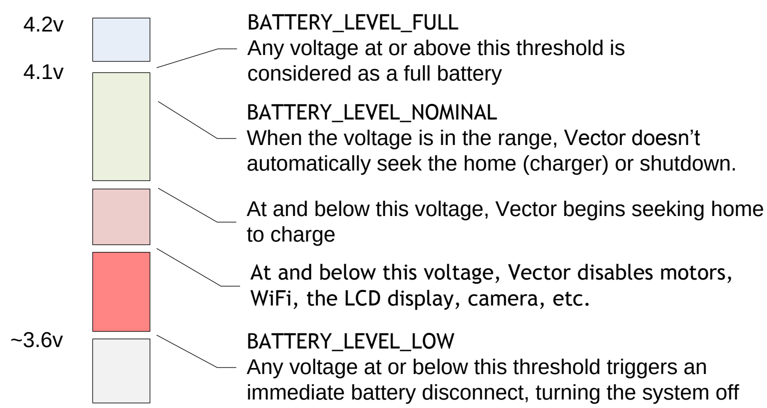



***Figure 24:** The battery thresholds*

The current battery level and voltage can be requested with the Battery State command (see Chapter 13, *Battery State*). The response will provide the current battery voltage, and interpreted level.

## 23.2. RESPONSES, SHEDDING LOAD / POWER SAVING EFFORTS

Vector's main (power-related) activity modes are:

- active, interacting with others
- calm, where primarily sitting still, waiting for assistance or stimulation
- sleeping

[24] The levels are from robot.py

Depending on the state of the battery – and charging – Vector may engage in behaviours that override others.

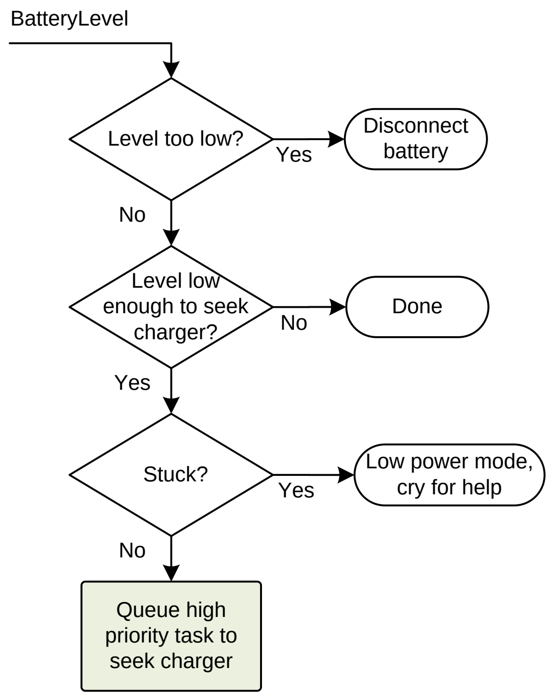



***Figure 25:*** *The response to battery level*

If Vector is unable to dock (or even locate a dock) he sheds load as he goes into a lower state:

- He no longer responds to his trigger word or communicates with WiFi servers
- He turns off camera and LCD; presumable the time of flight sensor as well.
- He reduces processing on the processor

### 23.2.1 Calm Power mode

Has a high-level power mode called "calm power mode." This mode "is generally when Vector is sleeping or charging."

Vector being in calm power mode is reported in the RobotStatus message in the status field. (See chapter 13 for details.) Vector is in a calm power model if the ROBOT_STATUS_CALM_POWER_MODE bit is set (in the status value).

## 24. CHARGING

Tracks whether is charging is in process, and how long. The software may estimate how long before charging is complete.[25]

The state of the charger is reported in the RobotStatus message in the status field. (See chapter 13 for details.) Vector is on the charger if the ROBOT_STATUS_IS_ON_CHARGER bit is set (in the status value), and charging if the ROBOT_STATUS_IS_CHARGING bit is set.

Additional information about the state of the charger can be requested with the Battery State command (see Chapter 13, *Battery State*). The response will provide flags indicating whether or

[25] It is possible, but unlikely, that the baseboard is acting as a coulomb counter by sampling the current across the resistor with low resistance. (The purpose of this resistor isn't known.)

not Vector is on the charger, and if it is charging. The response also provides a suggested amount of time to charge the batteries.

## CHAPTER 7

# Basic Inputs and Outputs

This chapter describes Vectors most basic input and output: his button, touch and LEDs:

- Touch and button input
- Backpack Lights control

## 25. BUTTON, TOUCH AND CLIFF SENSOR INPUT

Vector's backpack button used to wake (and silence) vector, or place him into recovery mode. Touch is used to pet Vector and provide him stimulation. 4 surface proximity IR sensors are used to detect cliffs and line edges. The responsibility for the button, touch, and proximity sensor input functions are divided across multiple processes and boards in Vector:

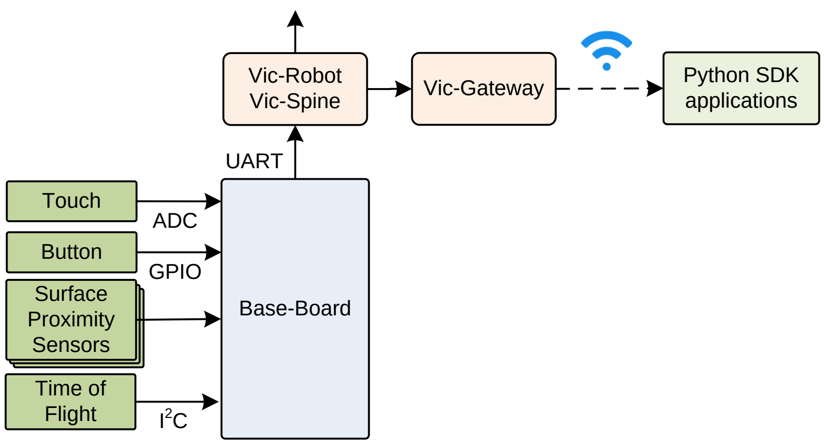



***Figure 26:*** *The touch and button input architecture*

The state of the inputs (button, touch, surface proximity and time of flight sensors) are reported in the `RobotStatus` message. (See chapter 13 for details.) The button state can be found in the `status` field. The button is pressed if the `ROBOT_STATUS_IS_BUTTON_PRESSED` bit is set (in the status value).

The touch sensor readings can be found in the `touch_data` field. The values indicate whether Vector is begging touched (e.g. petted).

The surface proximity sensors (aka "cliff sensors") are used to determine if there is a cliff, or potentially in the air. The individual sensor values are not accessible. The cliff detection state can be found in the `status` field. A cliff is presently detected if the `ROBOT_STATUS_CLIFF_DETECTED` bit is set (in the status value).

The time of flight reading is given in the prox_data field. This indicates whether there is a valid measurement, the distance to the object, and a metric that indicates how good the distance measurement is.

# 26. BACKPACK LIGHTS CONTROL

The backpack lights are used to show the state of the microphone, charging, WiFi and some other behaviours.

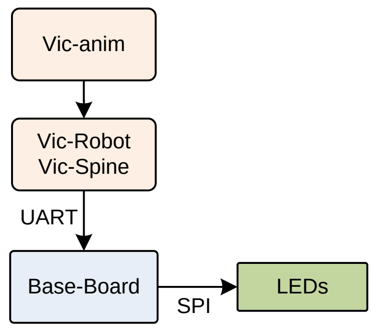



***Figure 27:*** *The backpack lights output architecture*

Vic-anim controls the backpack lights based on specifications in JSON files in

`/anki/data/assets/cozmo_resources/config/engine/lights/backpackLights/`

The path is hard coded into vic-anim. All of the JSON files have the same structure with the following fields:

***Table 14:*** *The Backpack LEDs JSON structure*

| Field | Type | Description |
|---|---|---|
| *offColors* | array of 3 colors | Each color corresponds to each of the 3 lower back pack lights |
| *offPeriod_ms* | array of 3 floats | The "off" duration (in ms) for each of the 3 back pack lights. This the duration to show the back pack light its corresponding "of" color (in offColors). |
| *offset* | array of 3 floats | always 0 |
| *onColors* | array of 3 colors | Each color corresponds to each of the 3 lower back pack lights |
| *onPeriod_ms* | array of 3 floats | The "on" duration (in ms) for each of the 3 back pack lights. This the duration to show the back pack light its corresponding "on" color (in onColors). |
| *transitionOffPeriod_ms* | array of 3 floats | The time (in ms) to transition from the on color to the off color. |
| *transitionOnPeriod_ms* | array of 3 floats | The time (in ms) to transition from the off color to the on color |

The colors are an array with 4 floats corresponding to red, green, blue, and alpha. The numbers are in the range of 0 to 1. Alpha is always 1. Note: red can be different for each of the LEDs, otherwise the blue and greens are the same for all three. The mid-range floating point values suggest that the baseboard PWM's the LEDs.

This structure is similar (not quite the same) as the one used with the companion cube lights.

CHAPTER 9

# Audio Input

This chapter describes the sound input and output system:

- The audio input
- The audio filtering, and triggering of the speech recognition

## 27. AUDIO INPUT

The audio input is [TBD] used to both give Vector verbal interaction, and to give him environmental stimulation:

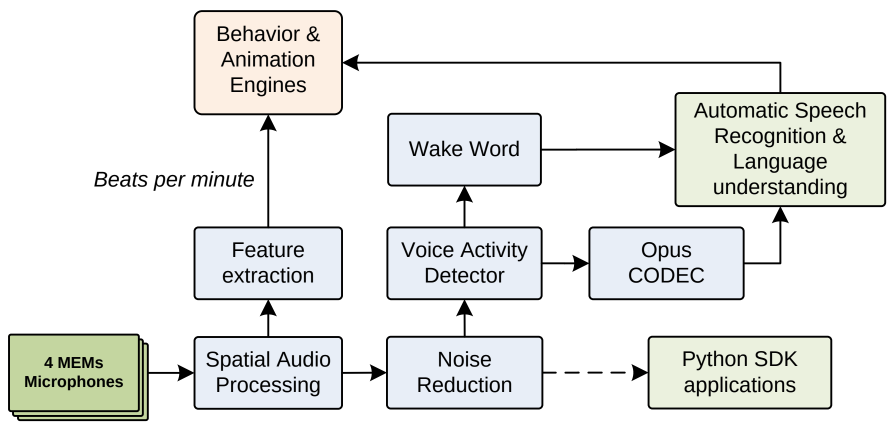



***Figure 28:** The audio input functional block diagram*

- Spatial audio processing localizes the sound of someone talking from the background music.
- The feature extraction detects the tempo of the music. If the tempo is right, Vector will dance to it. It also provides basic stimulation to Vector.
- Noise reduction makes for the best sound.
- Voice activity detector usually triggered off of the signal before the beam-forming.
- A wake word is used to engage the automatic speech recognition system. *Note: the wake word is also referred to as the trigger word.*
- The speech recognition system is on a remote server. The audio sent to the automatic speech recognition system is compressed to reduce data usage.

The responsibility for these functions is divided across multiple processes and boards in Vector:

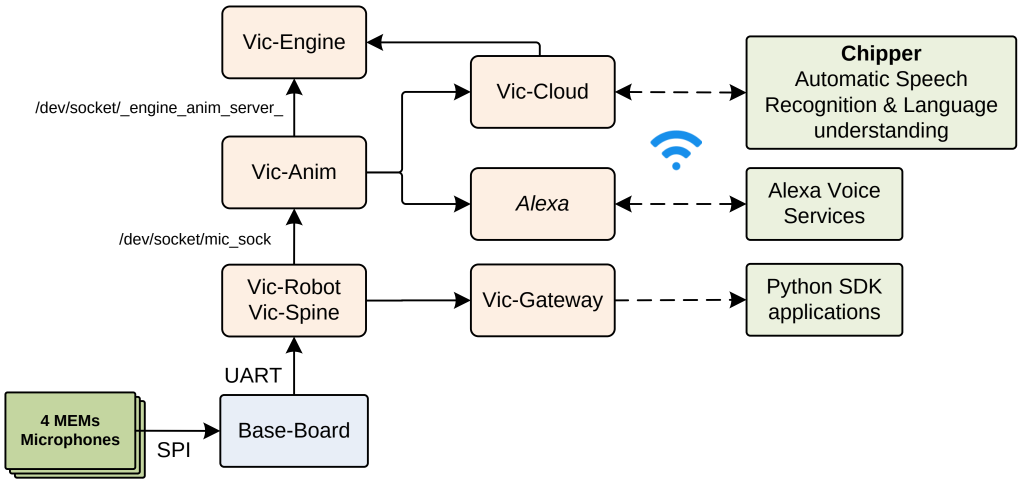



***Figure 29:*** *The audio input architecture*

*Note: providing the audio input to the SDK (via Vic-gateway) was never completed. It will be discussed based on what was laid out in the protobuf specification files.*

The audio processing blocks, except where otherwise discussed, are part of Vic-Anim. These blocks were implemented by Signal Essence, LLC. They probably consulted on the MEMs microphones and their configuration. The Qualcomm family includes software support for these tasks, as part of the Hexagon DSP SDK; it is not known how much of this Signal Essence took advantage off.

## 27.1. THE MICROPHONES

The microphone array is 4 MEMs microphones that sample the incoming sound and transfer the samples to base-board. The audio is sampled by the base-board at 15,625 samples/sec. The audio is transferred to the Vic-spine module (part of Vic-robot) in regular communication with the head-board. The audio samples are extracted and forward to the Vic-Anim process.

The audio samples, once received, are processed at 16,000 samples/sec. ("As a result, the pitch is altered by 2.4%.") The signal processing is done in chunks of 160 samples.

Note: The Customer Care Information Screen (CCIS) shows the microphones to be about 1024 when quiet. If this is center, the max would be 2048… or 11 bit. Probably is 12 bit.

## 27.2. SPATIAL AUDIO PROCESSING

The spatial audio processing is uses multiple microphones to pick-out the wanted signal and cancel out the unwanted. Note: The spatial audio processing is bypassed until voice activity has been detected.

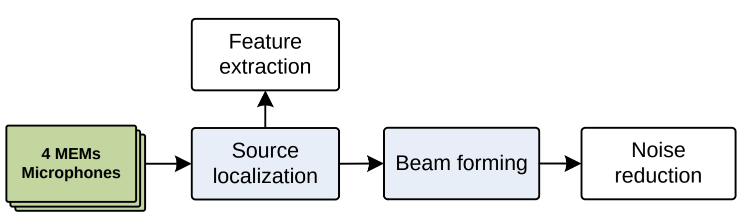



***Figure 30:*** *Typical spatial audio processing flow*

THE SOURCE LOCALIZATION estimates direction of arrival of the person talking.

BEAM-FORMING combines the multiple microphone inputs to cancels audio coming from other directions.

The output of this stage includes:

- A histogram of the directions that the sound(s) in this chunk of audio came from. There are 12 bins, each representing a 30° direction.
- The direction that is picked for the origin of the sound of interest
- A confidence value for that direction
- A measure of the background noise

## 27.3. NOISE REDUCTION

Noise reduction identifies and eliminates noise and echo in the audio input

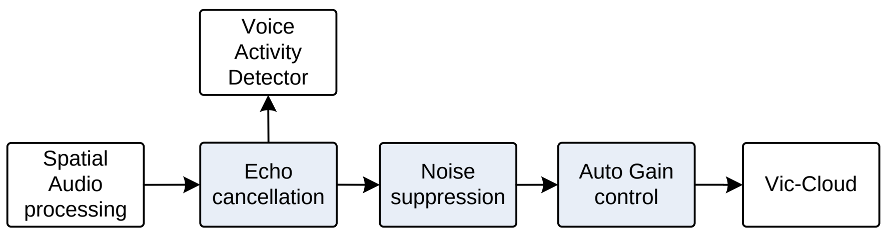



***Figure 31:*** *Typical audio noise reduction flow*

ACOUSTIC ECHO CANCELLATION cancels slightly delayed repetitions of a signal.

NOISE SUPPRESSION is used to eliminate noise.

The combination of spatial processing and noise reduction gives the cleanest sound (as compared with no noise reduction and/or no spatial processing).

## 27.4. VOICE ACTIVITY DETECTOR AND WAKE WORD

The voice activity detector is given cleaned up sound from multiple microphones without beam-forming. When it detects voice activity, then the spatial audio processing is fully enabled.[26]

The voice activity detector and the wake word are used so that downstream processing – the search for the wake word, and the automatic speech recognition system – are not engaged all the time. They are both expensive (in terms of power and data usage), and the speech recognition is prone to misunderstanding.

When the voice activity detector triggers – indicating that a person may be talking – the spatial audio processing is engaged (to improve the audio quality) and the audio signals are passed to the Wake Word Detector.

The detector for the "Hey, Vector" is provided by Sensory, Inc. Pryon, Inc provided the detector for "Alexa."[27] The recognition is locale dependent, detecting different wake words for German, etc. It may be possible to create other recognition files for other wake words.

[26] Vector's wake word detection, and speech recognition is pretty hit and miss. Signal Essence's demonstration videos show much better performance. The differences are they used more microphones and the spatial audio filtering in their demos.
[27] This appears to be standard for Alexa device SDKs.

When the “Hey, Vector” wake word is heard,

1. A connection (via Vic-Cloud) is made to the remote speech processing server for automatic speech recognition.
2. A “WakeWordBegin” event message is posted to Vic-Engine and Vic-Gateway. Vic-Gateway may forward the message on to a connected application.
3. An WakeWordEnd event message may be sent when the Vic-cloud has received an intent back, along with the intent JSON data structure (if any).

### 27.4.1 Wake work configuration file

The configuration file for the wake word is located at:

`/anki/data/assets/cozmo_resources/config/micData/micTriggerConfig.json`

This file has dictionary structure with the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *alexa_thf* | array of TBD | |
| *hey_vector_thf* | array of TBD | |

***Table 15:** The micTriggerConfig JSON structure*

Each TBD structure has the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *defaultModelType* | string | e.g. “size_500kb” |
| *locale* | string | The IETF language tag of the owner’s language preference – American English, UK English, Australian English, German, French, Japanese, etc.<br>default: “en-US” |
| *modelList* | array of TBD | |

***Table 16:** The TBD JSON structure*

## 27.5. CLOUD SPEECH RECOGNITION

The audio snippets are sent to a remote server known as “chipper” for processing.

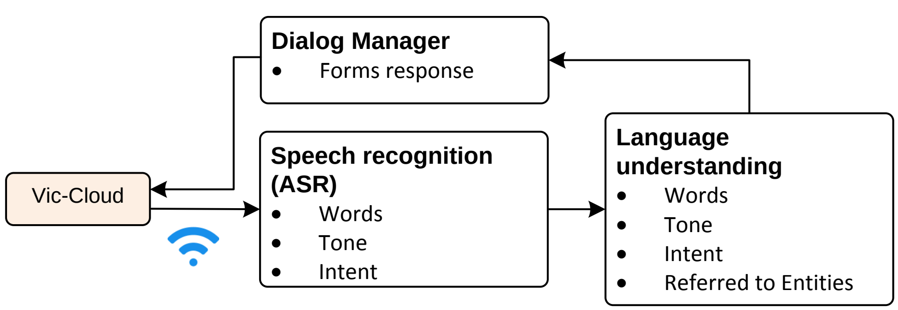



***Figure 32:** The speech recognition*

Chipper does speech recognition, and some language understanding. What the user says is mapped to a “user intent” – this may be a command, or a question to be answered. The intent also includes some supporting information – the colour to set the eyes to, for instance. Many of the phrase patterns and the intent they map to can be found in Appendix I. The intent may be further handled by Anki servers; the intent is eventually sent back to Vector.

The cloud (or application) intent is then mapped to the intent name used internally within Vector's engine. The configuration file holding this mapping is located at:

/anki/data/assets/cozmo_resources/config/engine/behaviorComponent/user_intent_map.json

The path is hard coded into libcozmo_engine.so. The file has the following structure:

***Table 17:*** *The user_intent_map JSON structure*

| Field | Type | Description |
|---|---|---|
| *user_intent_map* | array of X | A table that maps the intent received by the cloud or application to the intent name used internally |
| *unmatched_intent* | string | The intent to employ if cloud's intent cannot be found in the table above. Default: "unmatched_intent" |

Each of the mapping entries has the following structure:

***Table 18:*** *The intent mapping JSON structure*

| Field | Type | Description |
|---|---|---|
| *app_intent* | string | These have the pattern of starting with "intent_" followed by the same string as the user_intent. *Optional.* |
| *app_substitutions* | dictionary | A dictionary whose keys are the keys provided by the application's intent structure, and maps to the keys used internally. *Optional.* |
| *cloud_intent* | string | The intent name returned by the cloud. These have the pattern of starting with "intent_" followed by the same string as the user_intent. |
| *cloud_numerics* | array of strings | Names of keys that used as parameter values by the behaviour..?? *Optional.* |
| *cloud_substitutions* | dictionary | A dictionary whose keys are the keys provided by the cloud's intent structure, and maps to the keys used internally. *Optional.* |
| *feature_gate* | string | The name of the feature that must be enabled before this intent can be processed. *Optional.* |
| *test_parsing* | bool | Default: true. *Optional.* |
| *user_intent* | string | The name of the intent used internally within Vector's engine. |

The extra hoops to jump thru suggest that the development of the server, mobile application and Vector were not fully coordinated and needed this to bridge a gap.

## 27.6. CONNECTIONS WITH VIC-GATEWAY AND SDK ACCESS

An application has access to to the wake-word events and the received user intent events as they occur. When the wake word is said, a "WakeWord" event message is posted to Vic-Engine and Vic-Gateway, and to the connected application. When the response is received from the cloud, another "WakeWord" and UserIntent event messages are posted, along with the intent JSON data structure (if any). See Chapter 13, *Wake Word Event*

An external application can send an intent to Vector using the AppIntent command (see Chapter 13, *App Intent*.

### 27.6.1 Audio Stream

It is clear that Anki made provisions to connect the audio stream to Vic-Gateway (and potentially Vic-Cloud) but were unable to complete the features before they ceased operation. The SDK would have been able to:

- Enable and disable listening to the microphone(s)
- Select whether the audio would have the spatial audio filter and noise reduction processing done on it.
- Include the direction of sound information from the spatial audio processing (see section *27.2 Spatial audio processing*)
- 1600 audio samples; Note: this is 10x the chunk size of the internal processing size

## 28. REFERENCES AND RESOURCES

https://github.com/ARM-software/ML-KWS-for-MCU
A reference keyword listener for ARM microcontrollers.

**CHAPTER 10**

# Motion Sensing

This chapter describes Vector's motion sensing:

- Sensing motion and cliffs
- Detecting external events
- Measuring motion as feedback to motion control, and allow moving along paths in a smooth and controlled fashion

## 29. MOTION SENSING

Vector employs an IMU – an accelerometer and gyroscope in the same module – detect motion, such as falling or being bumped, as well as measuring the results of motor-driven motions.

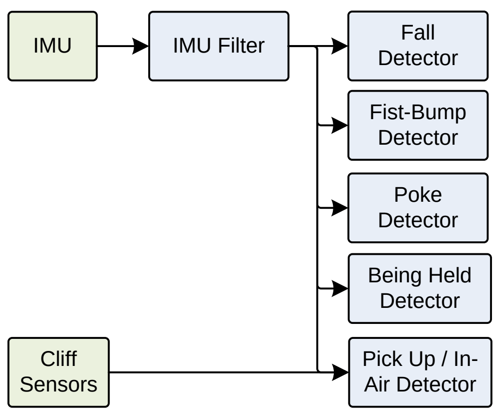



***Figure 33:*** *Motion sensing*

### 29.1. ACCELEROMETER AND GYROSCOPE

Neither the accelerometer nor a gyroscope by itself is sufficient. Accelerometers measure force along 3 (XYZ) axes. If there is no other motion, the accelerometer provides the orientation. Accelerometers cannot distinguish spins, and other rotation movements from other movements. Gyroscopes can measure rotations around those axes, but cannot measure linear motion along the axis. Gyroscopes also have a slight bias in the signal that they measure, giving the false signal that there is always some motion occurring.

The accelerometer and gyroscope signals are blended together to compensate and cancel each other's weaknesses out.

At start up, the gryroscope is calibrated.

## 29.2. TILTED HEAD

The IMU can also measure how tilted Vector's head is. The IMU is located in Vector's head. This presents a small extra step of processing for the software to accommodate the impact of the head. By combining the position & orientation of the IMU within the head, the current estimated angle of the head, the known joint that the head swivels on, and working backwards the IMU measures can be translated to body-centered measures.

## 29.3. SENSING MOTION

The IMU's primary function is detect motion – to help estimate the change in position, and orientation of Vector's body, and how fast it is moving.

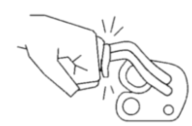
The IMU can be used to detect the angle of Vector's body. This is important, as the charging behaviour uses the tilt of the charging station ramp to know that it is in the right place.

## 29.4. SENSING INTERACTIONS

The IMU (with some help from the cliff sensors) is also used to sense interactions and other environmental events – such as being picked up or held by a person, being poked or given a fist bump, or falling.

The IMU can sense taps and pokes that provide some measurable signal, and may tile Vector… but also go away quickly and vector resumes his prior position.

Fist-bumps are like pokes, except that the lift has already been raised, and most of the motion will be predictable from receiving the bump on the lift.



Fall detector is similar. In free-fall, the force measured by the accelerometer gets very small. If Vector is tumbling, there is a lot of angular velocity that is taking Vector off of his driving surface.

Being picked up is distinct because of the direction of acceleration and previous orientation of Vector's body.

Being held is sensed, in part by first being picked up, and by motions in the IMU that indicate it is not on a solid, surface

# 30. REFERENCES AND RESOURCES

AdaFruit, https://github.com/adafruit/Adafruit_9DOF/blob/master/Adafruit_9DOF.cpp
An example of how accelerometer and gyroscope measurements are fused.

# PART III

# Communication

This part provides details of Vector's communication protocols. These chapters describe structure communication, the information that is exchange, its encoding, and the sequences needed to accomplish tasks. Other chapters will delve into the functional design that the communication provides interface to.

- COMMUNICATION. A look at Vector's communication stack.
- BLUETOOTH LE. The Bluetooth LE protocol that Vector responds to.
- SDK PROTOCOL. The HTTPS protocol that Vector responds to.
- CLOUD. A look at how Vector interacts with remote services

*drawing by Jesse Easley*

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

CHAPTER 11

# Communication

This chapter describes 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

## 31. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE

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

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

The communication stacks look something like:

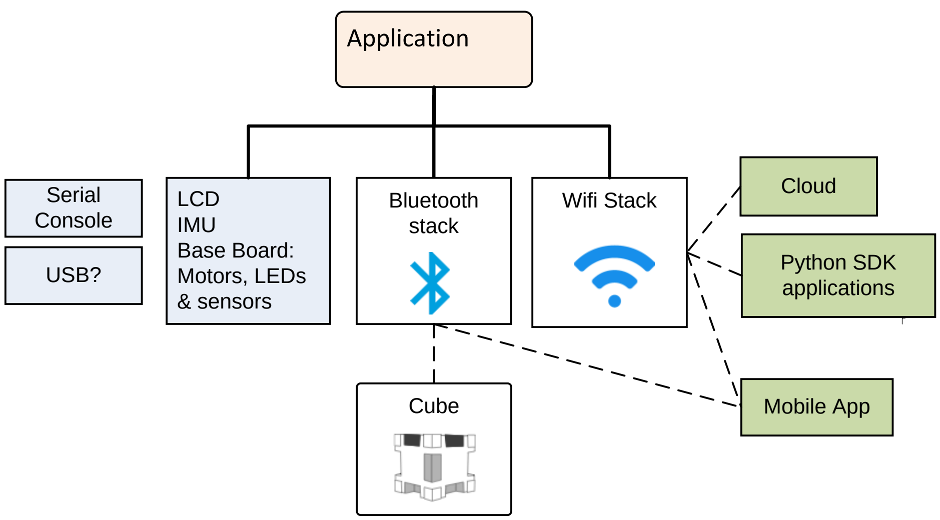



***Figure 34:*** *The overall communication infrastructure*

# 32. INTERNAL COMMUNICATION WITH PERIPHERALS

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

## 32.1. COMMUNICATION WITH THE BASE-BOARD

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

Data rate: 3 Mbits/sec[28]

### 32.1.1 Messages from the base-board to the head board

The base-board sends packets at regular intervals to the head-board. The frame of the message in

[Unknown byte] $AA_{16}$ 'B' '2' 'H' [16-bit packet type] [16-bit payload size] [payload bytes] [32-bit CRC]

(All multi-byte values are in little endian order.) The maximum packet size is 1280 bytes.

The packet type implies both the size of the payload, and the contents. If the packet type is not recognized, or the implied size does not match the passed payload size, the packet is considered in error.) The table below gives the different type codes:

***Table 19:*** *JSON structure*

| Packet type | Payload Size | Description |
|---|---|---|
| $6473_{16}$ | 0 | |
| $6b61_{16}$ | 4 | |
| $6466_{16}$ | 768 | The size of the message suggests that it holds 120 samples from one or two microphones (2 microphones × 2bytes/sample × 120 samples/microphone == 960 bytes) for the voice activity detection audio processing. |
| $6662_{16}$ | 4 | |
| $6675_{16}$ | 1028 | The size of the message suggests that it holds 120 samples from the microphones (4 microphones × 2bytes/sample × 120 samples/microphone == 960 bytes) for the spatial audio processing. |
| $736c_{16}$ | 16 | |
| $6d64_{16}$ | 0 | |
| $7276_{16}$ | 40 | |
| $7374_{16}$ | 0 | |
| $7878_{16}$ | 0 | |

The payload can contain (depending on the type of packet):

- The state of the backpack button
- The touch sensor voltage
- The microphone signals for all 4 microphones. (Most likely as 16 bits)

[28] Value from analysis of the RAMPOST program.

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

### 32.1.2 Messages from the head-board to the base-board

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

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

The head-board can update the firmware in the base-board, by putting into DFU (device firmware upgrade) mode and downloading the replacement firmware image.

## 32.2. SERIAL BOOT CONSOLE

The head-board employs a serial port to display kernel boot up and log messages. The parameters are 115200 bits/sec, 8 data bits no parity, 1 stop bit; the device file is `/dev/ttyHSL0`. This serial port is not bi-directional, and can not be used to login. *Melanie T*

## 32.3. 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 software during manufacture. *Melanie T*

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

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

# 33. BLUETOOTH LE

Bluetooth LE is used for two purposes:

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

Vector's Bluetooth LE stack looks like:

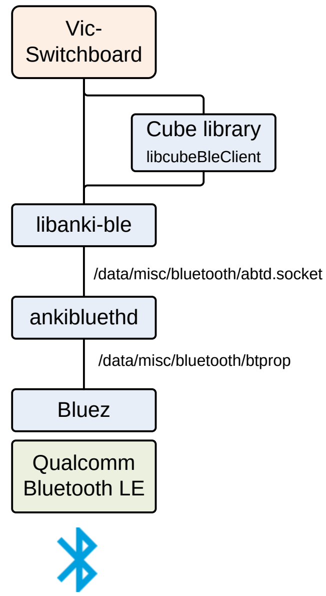



***Figure 35:*** *The Bluetooth LE stack*

The elements of the Bluetooth LE stack include:

***Table 20:*** *Elements of the Bluetooth LE stack*

| **Element** | **Description & Notes** |
|---|---|
| ankibluetoothd | A server daemon.  The application layer communicates with it over a socket; `/data/misc/bluetooth/abtd.socket` |
| BlueZ | Linux's official Bluetooth stack, including Bluetooth LE support. The Anki Bluetooth daemon interacts with it over a socket: `/data/misc/bluetooth/btprop` |
| bccmd | A Bluetooth core command |
| btmon | A command-line Bluetooth tool |
| libanki-ble.so | Communicates with Anki Bluetooth daemon probably serves both the external mobile application interface and communication with the companion cube. |
| libcubeBleClient.so[29] | A library to communicate with the companion cube, play animations on its LEDs, detect taps and the orientation of the cube. |
| viccubetool | Probably used to update the firmware in the Cube. |

# 34. WIFI

WiFi networking is used by Vector for five purposes:

1. WiFi is used to provide the access to the remote servers for Vector's speech recognition, natural language processing

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

2. WiFi is used to provide the access to the remote servers for software updates, and providing diagnostic logging and troubleshooting information to Anki
3. To provide time services to so that Vector knows the current time
4. To provide an interface, on the local network, that the mobile application can use to configure Vector, and change his settings.
5. To provide an interface, on the local network, that SDK applications can use to program Vector.

The WiFi network stack looks like:

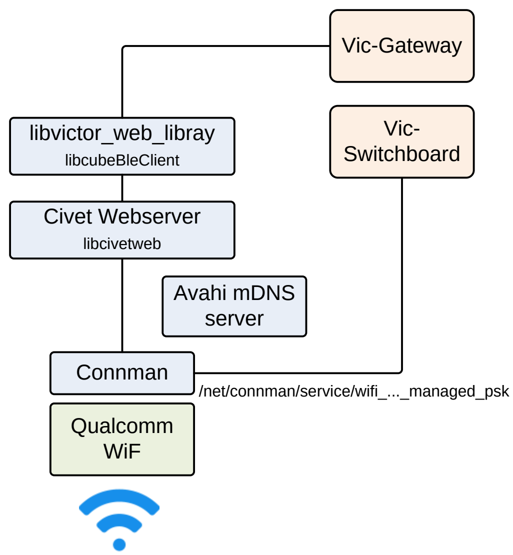



*Figure 36: The WiFi stack*

The elements of the stack include:[30]

*Table 21: Elements of the Bluetooth LE stack*

| Element | Description & Notes |
|---|---|
| avahi 0.6.31 | A mDNS server that registers Vector's robot name (with his network address) on the local network; |
| chronyd | Fetches the time from a network time server. |
| libcivetweb.so.1.9.1 | Embedded web server |
| libvictor_web_library.so | Anki Vector Web Services. |

## 34.1. FIREWALL

The network configuration includes a firewall set up with the usual configuration files:

```
/etc/iptables/iptables.rulesiptables
/etc/iptables/ip6tables.rulesiptables
```

[30] All of the software versions include an Anki webserver service systemd configuration file whose executable is missing. The most likely explanation is that early architecture (and possibly early versions) included this separate server, and that the systemd configuration file is an unnoticed remnant.

Is set to block incoming traffic (but not internal traffic), except for:

1. Responses to outgoing traffic
2. DHCP
3. TCP port 443 for vic-gateway
4. UDP port 5353 for mDNS (Avahi)
5. And the ping ICMP

The firewall does not block outgoing traffic

## 34.2. WIFI CONFIGURATION

The WiFi is configured by the Vic-switchboard over Bluetooth LE. The WiFi settings cannot be changed by the remote servers or thru the WiFi-based API; nor is information about the WiFi settings is not stored remotely.

The WiFi is managed by connman thru the Vic-Switchbox:

- To provide a list of WiFi SSIDs to the mobile app
- To allow the mobile app to select an SSID and provide a password to
- Tell it forget an SSID
- To place the WiFi into Access Point mode

# 35. COMMUNICATING WITH MOBILE APP AND SDK

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

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

## 35.1. CERTIFICATE BASED AUTHENTICATION

A *session token* is always provided by Anki servers.[31] It is passed to Vector to authenticate with him and create a client token. The session token is passed to Vector via the Bluetooth LE RTS protocol or the HTTPS-based SDK protocol; Vector will return a client token. The session token is single use only.

A *client token* is passed to Vector in each of the HTTPS-based SDK commands, and in the Bluetooth LE SDK Proxy commands. It is generated in one of two ways. One method is by the Bluetooth LE command (cloud session); the other is by send a User Authentication command (see Chapter 13 *User Authentication*). The client token should be saved indefinitely for future use. It is not clear if the client token can be shared between the two transport mechanisms.

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

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

Note: the certificates are invalidated and new ones are created when recovery-mode is used. Vector is assigned a new robot name as well.

The typical information embedded in a Vector certificate:

**Table 22:** Elements of a Vector certificate

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

The TLS certificates and signing keys are stored in the OEM partition, in the "cloud" folder:

**Table 23:** OEM cloud folder

| File | Description |
|---|---|
| *AnkiRobotDeviceCert.pem* | The |
| *AnkiRobotDeviceKeys.pem* | The |
| *Info$(serialNum}.json* | A configuration file that |
| *${serialNum}* | empty |

The Info${serialNum}.json file has the following structure:

**Table 24:** Cloud Info${serialNum} structure

| Field | Type | Description |
|---|---|---|
| *CertDigest* | base64 string | |
| *CertSignature* | base64 string | |
| *CertSignatureAlgorithm* | string | The name of openSSL signature algorithm to use, "sha256WithRSAEncryption" |
| *CommonName* | string | 'vic:' followed by the serial number. (This is also called the "thing id" in other structures. |
| *KeysDigest* | base64 string | |

# 36. REFERENCES & RESOURCES

PyCozmo.

https://github.com/zayfod/pycozmo/blob/master/docs/protocol.md
https://github.com/zayfod/pycozmo/blob/master/pycozmo/protocol_declaration.py

Vector has a couple UDP ports open internally; likely this is inherited from libcozmo_engine. The PyCozmo project has reverse engineered much of Cozmo's UDP protocol.

CHAPTER 12

# Bluetooth LE Communication Protocol

This chapter 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 F.*

## 37. COMMUNICATION PROTOCOL OVERVIEW

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

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

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

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

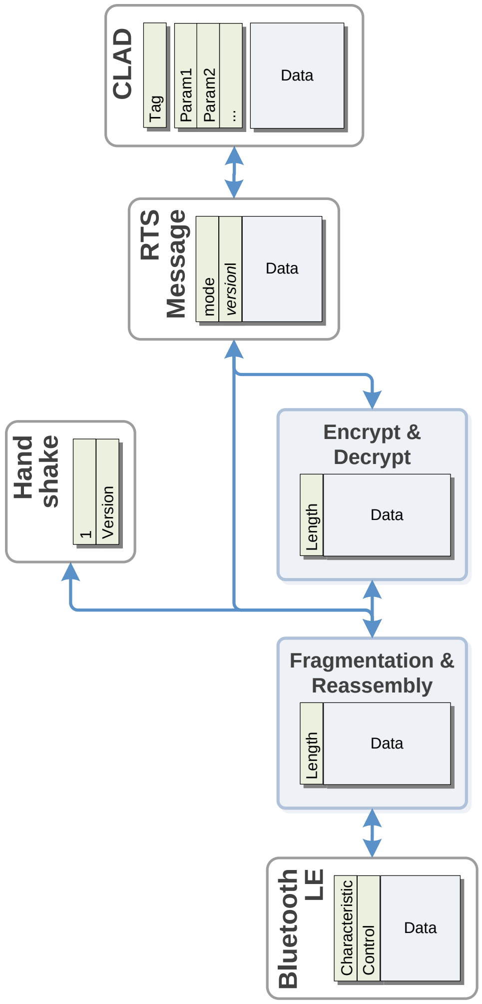



***Figure 37:*** *Overview of encryption and fragmentation stack*

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 F.

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,

## 37.1. SETTING UP THE COMMUNICATION CHANNEL

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

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

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

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

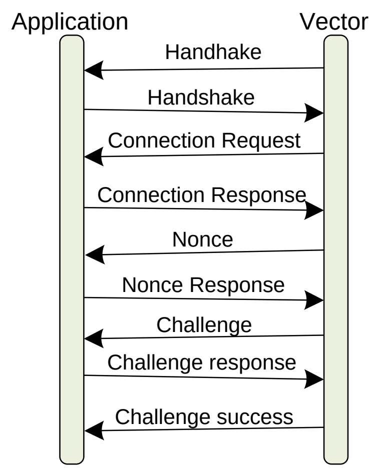



***Figure 38:*** *Sequence for initiating communication with Vector*

1. The application opens Bluetooth LE connection (retrieving the service and characteristics handles) and subscribes to the "read" characteristic (see Appendix F 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 25:*** *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 |

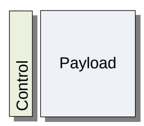
3. The application sends the handshake back
4. Then the Vector will send a *connection request,* consisting of the public key to use for the session. The application's response depends on whether this is a first-time pairing, or a reuse.
    a. First time pairing requires that Vector have already been placed into pairing mode prior to connecting to Vector. The application keys should be created (see section *37.3.1 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 *37.3.1 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.

## 37.2. FRAGMENTATION AND REASSEMBLY

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




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

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

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

Fragmenting reverses the process:

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

## 37.3. 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];
```

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

The variables mean:

| 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. |

***Table 26:** The encryption variables*

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

- First time pairing, and
- Reconnection

### 37.3.1 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);
```

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

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);
```

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

### 37.3.2 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?}

### 37.3.3 Encrypting and decryption messages

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

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

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

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

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

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

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

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

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

## 37.4. 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:

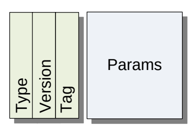



***Figure 39:*** *The format of an RTS frame*

- 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.

## 37.5. FETCHING A LOG

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

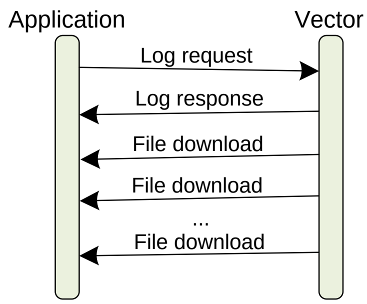



***Figure 40:*** *Sequence for initiating communication with Vector*

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

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

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

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

# 38. 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 27:* Summary of the commands

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

## 38.1. APPLICATION CONNECTION ID

?

### 38.1.1 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 28:*** *Parameters for Application Connection Id request*

### 38.1.2 Response

There is no response.

## 38.2. CANCEL PAIRING

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

### 38.2.1 Request

The command has no parameters.

### 38.2.2 Response

There is no response.

## 38.3. CHALLENGE

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

### 38.3.1 Request

The parameters of the request body are:

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

***Table 29:*** *Parameters for challenge request*

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

### 38.3.2 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 30:*** *Parameters for challenge response*

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

## 38.4. CHALLENGE SUCCESS

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

### 38.4.1 Request

The command has no parameters.

### 38.4.2 Response

There is no response.

## 38.5. CLOUD SESSION

This command is used to request a cloud session.

### 38.5.1 Command

The parameters of the request body are:

| Offset | Size | Type | Parameter | Description |
|---|---|---|---|---|
| 0 | 2 | uint16_t | *session token length* | The number of bytes in the session token; may be 0 |
| 2 | varies | uint8_t | *session token* | The session token, as received from the cloud server.[32] |
| | 1 | uint8_t | *client name length* | The number of bytes in the client name string; may be 0 version >= 5 |
| | *varies* | uint8_t[] | *client name* | The client name string. Informational only. The mobile app uses the name of the mobile device. version >= 5 |
| | 1 | uint8_t | *application id length* | The number of bytes in the application id string; may be 0; version >= 5 |
| | varies | uint8_t[] | *application id* | The application id. Informational only. The mobile uses "companion-app". version >= 5 |

***Table 31:*** *Parameters for Cloud Session request*

### 38.5.2 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 33: Cloud status enumeration* |
| 2 | 1 | uint16_t | *client token GUID length* | The number of bytes in the client token GUID; may be 0 |
| | varies | uint8_t[] | *client token GUID* | The client token GUID. The client token GUID should be saved for future use. |

***Table 32:*** *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 |

***Table 33:*** *Cloud status enumeration*

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

| | |
|---|---|
| 5 | authorized as secondary |
| 6 | reauthorization |

## 38.6. CONNECT

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

### 38.6.1 Request

The parameters of the request body are:

*Table 34:* Parameters for Connection request

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

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

### 38.6.2 Response

The parameters for the connection response message:

*Table 35:* Parameters for Connection Response

| Offset | Size | Type | Parameter | Description |
|---|---|---|---|---|
| 0 | 1 | uint8_t | *connectionType* | See *Table 36: Connection types enumeration* |
| 1 | 32 | uint8_t[32] | *publicKey* | The public key to use for the connection |

The connection types are:

*Table 36:* Connection types enumeration

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

The application sends the response, with its publicKey (see section *37.3 Encryption support*). A "first time pairing" connection type will cause Vector to display a pin code on the screen

If a first time pairing response is sent:

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

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

## 38.7. 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.

### 38.7.1 Request

The command has no parameters.

### 38.7.2 Response

There is no response.

## 38.8. FILE DOWNLOAD

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

### 38.8.1 Request

There is no direct request.

### 38.8.2 Response

The parameters of the response 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 | uint16_t | *length* | The number of bytes to follow (can be 0) |
| | varies | uint8_t[length] | *bytes* | The bytes of this file chunk |

***Table 37:*** *Parameters for File Download request*

## 38.9. LOG

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

### 38.9.1 Request

The parameters of the request body are:

***Table 38:*** *Parameters for Log request*

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

Each filter entry has the following structure:

***Table 39:*** *Log filter*

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

### 38.9.2 Response

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

The parameters for the response message:

***Table 40:*** *Parameters for Log Response*

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

## 38.10. NONCE

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

### 38.10.1 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 41:*** *Parameters for Nonce request*

### 38.10.2 Response

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

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

***Table 42:*** *Parameters for Nonce response*

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

## 38.11. OTA UPDATE

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

### 38.11.1 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 43:*** *Parameters for OTA request*

### 38.11.2 Response

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

The parameters for the response message:

| Offset | Size | Type | Parameter | Description |
|---|---|---|---|---|
| 0 | 1 | uint8_t | *status* | See *Table 204: OTA update-engine status* |
| 1 | 8 | uint64_t | *current* | The number of bytes downloaded |
| 9 | 8 | uint64_t | *expected* | The number of bytes expected to be downloaded |

***Table 44:*** *Parameters for OTA Response*

The OTA status codes are:

| Status | Meaning |
|---|---|
| 0 | idle |
| 1 | unknown |
| 2 | in progress |
| 3 | complete |
| 4 | rebooting |
| 5 | error |
| 200… | Status codes from the update-engine. See Appendix D, *Table 204: OTA update-engine status codes* for these update-engine status codes. |

***Table 45:*** *OTA status enumeration*

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

## 38.12. RESPONSE

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

***Table 46:*** *Parameters for Response*

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

## 38.13. SDK PROXY

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

### 38.13.1 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 47:*** *Parameters for the SDK proxy request*

### 38.13.2 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 48:*** *Parameters for the SDK proxy Response*

## 38.14. SSH

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

### 38.14.1 Request

The parameters for the request message:

| 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). |

***Table 49:*** *Parameters for SSH request*

Each authorization key has the following structure:

| 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 |

***Table 50:*** *SSH authorization key*

### 38.14.2 Response

The response has no parameters.

## 38.15. STATUS

This command is used to request basic info from Vector.

### 38.15.1 Request

The request has no parameters.

### 38.15.2 Response

The parameters for the response message:

***Table 51:*** *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 52: WiFi state enumeration* |
| | 1 | uint8_t | *access point* | 0 not acting as an access point, otherwise acting as an access point |
| | 1 | uint8_t | *Bluetooth LE state* | 0 if the Bluetooth |
| | 1 | uint8_t | *Battery state* | |
| | 1 | uint8_t | *version length* | The number of bytes in the version string; may be 0 version >= 2 |
| | *varies* | uint8_t [version length] | *version* | The version string; version >= 2 |
| | 1 | uint8_t | *ESN length* | The number of bytes in the ESN string; may be 0 version >= 4 |
| | *varies* | uint8_t[ESN length] | *ESN* | The *electronic serial number* string; version >= 4 |
| | 1 | uint8_t | *OTA in progress* | 0 over the air update not in progress, otherwise in process of over the air update; version >= 2 |
| | 1 | uint8_t | *has owner* | 0 does not have owner, otherwise has owner; version >= 3 |
| | 1 | uint8_t | *cloud authorized* | 0 is not cloud authorized, otherwise is cloud authorized; version >= 5 |

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

The WiFi states are:

***Table 52:*** *WiFi state enumeration*

| Index | Meaning |
|---|---|
| 0 | Unknown |
| 1 | Online |
| 2 | Connected |
| 3 | Disconnected |

## 38.16. WIFI ACCESS POINT

This command is used to request that the Vector act as a WiFi access point. This command requires that a "cloud session" have been successfully started first (see section *38.5 Cloud session*).

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

### 38.16.1 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 53:*** *Parameters for WiFi Access Point request*

### 38.16.2 Response

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

The parameters for the response message:

| 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 54:*** *Parameters for WiFi Access Point Response*

## 38.17. WIFI CONNECT

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

### 38.17.1 Request

The parameters for the request message:

***Table 55:*** *Parameters for WiFi Connect request*

| Offset | Size | Type | Parameter | Description |
|---|---|---|---|---|
| 0 | 1 | uint8_t | *SSID length* | The number of bytes in the SSID string; may be 0 |
| 1 | *varies* | uint8_t[SSID length] | *SSID* | The WiFi SSID (hex string) |
| | 1 | uint8_t | *password length* | The number of bytes in the password string; may be 0 |
| | *varies* | uint8_t [password length] | *password* | The WiFi password |
| | 1 | uint8_t | *timeout* | How long to given the connect attempt to succeed. |
| | 1 | uint8_t | *auth type* | The type of authentication to employ; see *Table 56: WiFi authentication types enumeration* |
| | 1 | uint8_t | *hidden* | 0 the access point is not hidden; 1 it is hidden |

The WiFi authentication types are:

***Table 56:*** *WiFi authentication types enumeration*

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

### 38.17.2 Response

The parameters for the response message:

***Table 57:*** *Parameters for WiFi Connect command*

| Offset | Size | Type | Parameter | Description |
|---|---|---|---|---|
| 0 | 1 | uint8_t | *SSID length* | The length of the SSID that was deleted; may be 0 |
| 1 | *varies* | uint8_t[SSID length] | *SSID* | The SSID (hex string) that was deleted |
| | 1 | uint8_t | *WiFi state* | See *Table 52: WiFi state enumeration* |
| | 1 | uint8_t | *connect result* | version >= 3 |

## 38.18. WIFI FORGET

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

### 38.18.1 Request

The parameters for the request message:

**Table 58:** *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 |

### 38.18.2 Response

The parameters for the response message:

**Table 59:** *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 |

## 38.19. WIFI IP ADDRESS

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

### 38.19.1 Request

The request has no parameters

### 38.19.2 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 60:*** *Parameters for WiFi IP Address response*

## 38.20. WIFI SCAN

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

### 38.20.1 Request

The command has no parameters.

### 38.20.2 Response

The parameters for the response message:

***Table 61:*** *Parameters for WiFi scan response*

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

Each access point has the following structure:

***Table 62:*** *Parameters access point structure*

| Offset | Size | Type | Parameter | Description |
|---|---|---|---|---|
| 0 | 1 | uint8_t | *auth type* | The type of authentication to employ; see *Table 56: 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 |

## CHAPTER 13

# The HTTPS based API

This chapter describes the communication with Vector over the local HTTPS. This chapter provides supplemental information not available at: https://developer.anki.com/vector/docs/proto.html

The Anki documentation include descriptions of the following types:

- ActionResult
- ProxData
- ResponseStatus
- TouchData

## 39. OVERVIEW OF THE SDK HTTPS API

The descriptions below[33] give the JSON keys, value format. It is implemented as gRPC/protobufs interaction over HTTP. (Anki has frequently said that the SDK included code (as python) with the protobuf spec so that others could use their own preferred language.) Each command is requested by POST-ing the request structure to the given relative URL (relative to Vector's address or local network name) and interpreting the returned body as the response structure.

The HTTPS header should include

- Bearer *BASE64KEY*
- Content-Type: application/json

(The JSON request is posted in the body)

### 39.1. MESSAGE GROUPINGS IN THIS CHAPTER

The major groups of messages here are:

- Accessories and custom objects
- Actions
- Alexa configuration
- Attention Transfer
- Audio
- Authentication

[33] The protocol was specified in Google Protobuf.

- Battery
- Camera
- Charger
- Connection, including checking the connection with the cloud
- Cube
- Display
- Event stream
- Faces (of people, not Vector’s face)
- Features and entitlements
- Interactions with objects (outside of the cube)
- JDocs, the JSON document storage interface
- Map and Navigation
- Miscellaneous items
- Motion Control
- Motion Sensing
- Onboarding
- Photos
- Settings and Preferences
- Software Updates, used to update Vector’s software – operating system, applications, assets, etc.

# 40. AUDIO

## 40.1. ENUMERATIONS

### 40.1.1 MasterVolumeLevel

The MasterVolumeLevel is used to control the volume of audio played by Vector, including text to speech. It is used in the MasterVolumeLevelRequest. The enumeration has the following named values:

| Name | Value | Description |
|---|---|---|
| *VOLUME_LOW* | 0 | |
| *VOLUME_MEDIUM_LOW* | 1 | |
| *VOLUME_MEDIUM* | 2 | |
| *VOLUME_MEDIUM_HIGH* | 3 | |
| *VOLUME_HIGH* | 4 | |

***Table 63:*** *MasterVolumeLevel Enumeration*

### 40.1.2 UtteranceState

The UtteranceState is used to control the volume of audio played by Vector, including text to speech. It is used in the MasterVolumeLevelRequest. The enumeration has the following named values:

| Name | Value | Description |
|---|---|---|
| *INVALID* | 0 | |
| *GENERATING* | 1 | Vector is generating the audio and other animation for the text to speech. |
| *READY* | 2 | Vector has completed generating the audio and animation. |
| *PLAYING* | 3 | Vector is playing the speech and related animation. |
| *FINISH* | 4 | Vector has finished playing the audio and animation. |

***Table 64:*** *UtteranceState Enumeration*

## 40.2. MASTER VOLUME

This command is used to request the state of Vector's battery and the cube battery. The state includes its voltage, and whether Vector is charging.

### 40.2.1 Request

The MasterVolumeResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 65:*** *MasterVolumeResponse JSON structure*

### 40.2.2 Response

The MasterVolumeResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 66:*** *MasterVolumeResponse JSON structure*

## 40.3. SAY TEXT

This command is used to request the state of Vector's battery and the cube battery. The state includes its voltage, and whether Vector is charging.

Post: "/v1/say_text"

### 40.3.1 Request

The SayTextRequest structure has the following fields:

*Table 67: SayTextRequest JSON structure*

| Field | Type | Units | Description |
|---|---|---|---|
| *duration_scalar* | float | *ratio* | This controls the speed at which Vector speaks. 1.0 is normal rate, less than 1 increases the speed (e.g. 0.8 causes Vector to speak in just 80% of the usual time), and a value larger than one slows the speed (e.g. 1.2 causes Vector to take 120% of the usual time to speak). Default: 1.0 |
| *text* | string | | The text (the words) that Vector should say. |
| *use_vector_voice* | bool | | True if the text should be spoken in "Vector's robot voice; otherwise, he uses a generic human male voice." |

### 40.3.2 Response

The SayTextResponse structure has the following fields:

*Table 68: SayTextResponse JSON structure*

| Field | Type | Description |
|---|---|---|
| *state* | UtteranceState | Where in the speaking process Vector is currently. |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

Note: all quotes above are from the python SDK. TBD: are multiple responses sent as the task progresses?

## 40.4. APP INTENT

This command is allows the mobile application or SDK application to send an intent to Vector. See also *User Intent,* and *Wake Word*

Post: "/v1/app_intent"

### 40.4.1 Request

The AppIntentRequest structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *intent* | string | The name of the intent (see below) to request |
| *param* | string | The parameters as a JSON formatted string. This can be empty if the intent does not require any additional information. |

***Table 69:*** *AppRequest JSON structure*

Vector (probably) will only honor the following intents:

- explore_start
- intent_clock_settimer
- intent_imperative_come
- intent_imperative_dance
- intent_imperative_fetchcube
- intent_imperative_findcube
- intent_imperative_lookatme
- intent_imperative_lookoverthere
- intent_imperative_quiet
- intent_imperative_shutup
- intent_meet_victor
- intent_message_playmessage
- intent_message_recordmessage
- intent_names_ ask
- intent_play_specific
- intent_system_charger
- intent_system_sleep
- knowledge
- knowledge_question
- knowledge_response
- _unknown

### 40.4.2 Response

The AppIntentResponse has the following fields:

| Field | Type | Description |
| --- | --- | --- |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 70:*** *AppIntentResponse JSON structure*

## 40.5. EVENTS

The following events are sent in the Event message.

### 40.5.1 User Intent Event

This event is sent when an event is received (from the cloud). The UserIntent structure has the following fields:

| Field | Type | Description |
| --- | --- | --- |
| *intent_id*[34] | uint32 | The identifier for the intent. See Appendix I *Table 219: The "Hey Vector" phrases* for an enumeration. |
| *json_data* | string | The parameters as a JSON formatted string. This may be empty if there is not additional information. |

***Table 71:*** *UserIntent JSON structure*

### 40.5.2 Wake Word Event

This event is sent when an the wake word is heard, and then when the cloud response is received. The WakeWord structure has the following fields, only one is present at any time:

| Field | Type | Description |
| --- | --- | --- |
| *wake_word_begin* | WakeWordBegin | This is sent when the wake word is heard. The structure has no contents. |
| *wake_word_end* | WakeWordEnd | This is sent when the response (and potential intent) is received from the cloud. TBD: is this sent before, after or instead of UserIntent? |

***Table 72:*** *WakeWord JSON structure*

The WakeWordEnd structure has the following fields:

| Field | Type | Description |
| --- | --- | --- |
| *intent_heard* | bool | True if a sentence was recognized with an associated intent; false otherwise. |
| *intent_json* | string | The intent and parameters as a JSON formatted string. This may be empty if an intent was not heard. |

***Table 73:*** *WakeWordEnd JSON structure*

[34] The use of an enumeration rather than a string is unusual here, and seems limiting.

# 41. AUTHENTICATION

## 41.1. ENUMERATIONS

### 41.1.1 Code

**Table 74:** *Code Enumeration*

| Name | Value | Description |
|---|---|---|
| *AUTHORIZED* | 1 | |
| *UNAUTHORIZED* | 0 | |

## 41.2. USER AUTHENTICATION

This command is used to authenticate

Post: "/v1/user_authentication"

### 41.2.1 Request

The UserAuthenticationRequest has the following fields:

**Table 75:** *UserAuthenticationRequest JSON structure*

| Field | Type | Description |
|---|---|---|
| *client_name* | bytes | |
| *user_session_id* | bytes | |

### 41.2.2 Response

The UserAuthenticationResponse has the following fields:

**Table 76:** *UserAuthenticationResponse JSON structure*

| Field | Type | Description |
|---|---|---|
| *client_token_guid* | bytes | The token bytes to be included in subsequent HTTPS postings. This token should be saved for future use. |
| *code* | Code | The result of the authentication request |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

# 42. BATTERY

## 42.1. ENUMERATIONS

The BatteryLevel enumeration is located in Chapter 6, Power Management, *Table 13: BatteryLevel codes as they apply to Vector*

## 42.2. BATTERY STATE

This command is used to request the state of Vector's battery and the cube battery. The state includes its voltage, and whether Vector is charging.

Post: "/v1/battery_state"

### 42.2.1 Request

*No parameters*

### 42.2.2 Response

The BatteryStateResponse structure has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *battery_level* | BatteryLevel | | The interpretation of the battery level. |
| *battery_volts* | float | *volts* | The battery voltage. |
| *cube_battery* | CubeBattery | | The status of the companion Cube's battery. |
| *is_on_charger_platform* | bool | | True if Vector is on his "home," aka charger. |
| *is_charging* | bool | | True if Vector is charging, false otherwise. |
| *status* | ResponseStatus | | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |
| *suggested_charger_sec* | float | *seconds* | Suggested amount of time to charge. |

***Table 77:*** *BatteryStateResponse JSON structure*

# 43. CONNECTION

## 43.1. ENUMERATIONS

### 43.1.1 ConnectionCode

The ConnectionCode is used to indicate whether the cloud is available. It is used in the response to the CheckCloudConnectionRequest command. The ConnectionCode enumeration has the following named values:

| Name | Value | Description |
|---|---|---|
| *AVAILABLE* | 1 | The cloud is connected, and has authenticated successfully |
| *BAD_CONNECTIVITY* | 2 | The internet or servers are down |
| *FAILED_AUTH* | 4 | The cloud connection has failed due to an authentication issue |
| *FAILED_TLS* | 3 | The cloud connection has failed due to [TLS certificate?] issue |
| *UNKNOWN* | 0 | There is an error connecting to the cloud, but the reason is unknown |

***Table 78:*** *ConnectionCode Enumeration*

## 43.2. CHECK CLOUD CONNECTION

Post: "/v1/check_cloud_connection"

### 43.2.1 Request

The CheckCloudRequest has no fields.

### 43.2.2 Response

The CheckCloudResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *code* | ConnectionCode | Whether the cloud is available, or the relevant connection error |
| *expected_packets* | int32 | |
| *num_packets* | int32 | |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |
| *status_message* | string | |

***Table 79:*** *CheckCloudResponse JSON structure*

# 44. CUBE

Cube Battery structure and enumeration

## 44.1. ENUMERATIONS

### 44.1.1 CubeBatteryLevel

The CubeBatteryLevel enumeration is used to categorize the condition of the Cube battery:

| Name | Value | Description |
|---|---|---|
| *BATTERY_LEVEL_LOW* | 0 | The Cube battery is 1.1V or less. |
| *BATTERY_LEVEL_NORMAL* | 1 | The Cube battery is at normal operating levels, i.e. >1.1v |

***Table 80:*** *CubeBatteryLevel codes[35] as they apply to Vector*

## 44.2. STRUCTURES

### 44.2.1 CubeBattery

The CubeBattery structure has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *battery_volts* | float | *volts* | The battery voltage. |
| *factory_id* | string | | The text string reported by the cube via Bluetooth LE. |
| *level* | CubeBatteryLevel | | The interpretation of the battery level. |
| *time_since_last_reading_sec* | float | *seconds* | The number of seconds that have elapsed since the last Bluetooth LE message from the cube with a battery level measure. |

***Table 81:*** *CubeBattery JSON structure*

[35] The levels are from robot.py

# 45. FACES

This section describes the commands and queries related to Vector's detection of faces, and managing what he knows about them. For a description of the facial detection and recognition process, see Chapter 15 *Face and Facial features recognition.*

Note: an int32 identifier is used to distinguish between faces that are seen. Each face will have a separate identifier. A positive identifier is used for a face that is known (recognized). This value will be the same when the face disappears and reappears later; the value likely persists across reboots. A negative identifier is used for face that is not recognized; as unknown faces appear and disappear they may be assigned different subsequent negative numbers. If a face becomes recognized, a RobotChangedObservedFaceID event will be sent, along with a change in identifier used.

## 45.1. ENUMERATIONS

### 45.1.1 FaceEnrollmentResult

The FaceEntrollmentResult is used to represent the success of associating a face with a name, or an reason code if there was an error. The enumeration has the following named values:

***Table 82:*** *FacialExpression Enumeration*

| Name | Value | Description |
|---|---|---|
| *SUCCESS* | 0 | A face was seen, its facial signature and associated name were successfully saved. |
| *SAW_WRONG_FACE* | 1 | |
| *SAW_MULTIPLE_FACES* | 2 | Too many faces were seen, and Vector did not know which one to associate with the name. |
| *TIMED_OUT* | 3 | |
| *SAVED_FAILED* | 4 | There was an error saving the facial signature and associated name to non-volatile storage. |
| *INCOMPLETE* | 5 | |
| *CANCELLED* | 6 | See Cancel Face Enrollment. |
| *NAME_IN_USE* | 7 | |
| *NAMED_STORAGE_FULL* | 8 | There was no more room in the non-volatile storage to hold another facial signature and associated name. |
| *UNKOWN_FAILURE* | 9 | |

### 45.1.2 FacialExpression

The FacialExpression is used to estimate the emotion expressed by each face that vector sees. The enumeration has the following named values:

| Name | Value | Description |
|---|---|---|
| *EXPRESSION_UNKNOWN* | 0 | The facial expression could not be estimated. Note: this could be because the facial expression estimation is disabled. |
| *EXPRESSION_NEUTRAL* | 1 | The face does not appear to have any particular expression. |
| *EXPRESSION_HAPPINESS* | 2 | The face appears to be happy |
| *EXPRESSION_SURPRISE* | 3 | The face appears to be surprised. |
| *EXPRESSION_ANGER* | 4 | The face appears |
| *EXPRESSION_SADNESS* | 5 | The face appears to be sad. |

***Table 83:*** *FacialExpression Enumeration*

## 45.2. STRUCTURES

### 45.2.1 RobotRenamedEnrolledFace

The RobotRenamedEnrolledFace structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *face_id* | int32 | The identifier code for the face. |
| *name* | string | The name now associated with the face. |

***Table 84:*** *RobotRenamedEnrolledFace JSON structure*

## 45.3. EVENTS

### 45.3.1 FaceEnrollmentComplete

The FaceEnrollmentComplete structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *face_id* | int32 | The identifier code for the face. |
| *name* | string | The name associated with the face. |
| *result* | FaceEnrollmentResult | Whether or not the face enrollment was successful; an error code if not. |

***Table 85:*** *FaceEnrollmentComplete JSON structure*

### 45.3.2 Meet Victor Face Scan Started

The MeetVictorFaceScanStarted structure has no fields.

### 45.3.3 Meet Victor Face Scan Complete

The MeetVictorFaceScanComplete structure has no fields.

### 45.3.4 RobotChangedObservedFaceID

This event occurs when a tracked (but not yet recognized) face is recognized and receives a positive ID. This happens when Vector's view of the face improves. This event can also occur "when face records get merged" "(on realization that 2 faces are actually the same)."

The FeatureFlagResponse type has the following fields:

*Table 86: RobotChangedObservedFaceID JSON structure*

| Field | Type | Description |
|---|---|---|
| *new_id* | int32 | The new identifier code for the face that has been recognized. |
| *old_id* | int32 | The identifier code that was used for the face until now. Probably negative |

### 45.3.5 RobotObservedFace

The RobotObservedFace event is sent when faces are observed within the field of view. This event is only sent if face detection is enabled. This structure has the following fields:

*Table 87: RobotObservedFace JSON structure*

| Field | Type | Description |
|---|---|---|
| *face_id* | int32 | The identifier code for the face; negative if the face is not recognized, positive if it has been recognized. |
| *expression* | FacialExpression | The estimated facial expression seen on the face. |
| *expression_values* | uint32[] | An array that represents the histogram of confidence scores in each individual expression. If the expression is not known (e.g. expression estimation is disabled), the array will be all zeros. Otherwise, will sum to 100. |
| *img_rect* | CladRect | The area within Vector's camera view that holds the face. |
| *name* | string | The name associated with the face (if recognized). Empty if a name is not known. |
| *pose* | PoseStruct | The position and orientation of the face. |
| *left_eye* | CladPoint[] | A polygon outlining the left eye, with respect to the image rectangle. |
| *mouth* | CladPoint[] | A polygon outlining the mouth; the coordinates are in the camera image. |
| *nose* | CladPoint[] | A polygon outlining the nose; the coordinates are in the camera image. |
| *right_eye* | CladPoint[] | A polygon outlining the right eye; the coordinates are in the camera image. |
| *timestamp* | uint32 | The time that the most recent facial information was obtained. |

## 45.4. CANCEL FACE ENROLLMENT

Cancels the request to look for a face and associate the face with a name.

post: "/v1/cancel_face_enrollment"

### 45.4.1 Request

The CancelFaceEnrollmentRequest structure has no fields.

### 45.4.2 Response

The CancelFaceEnrollmentResponse has the following fields:

***Table 88:*** *CancelFaceEnrollmentResponse JSON structure*

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

## 45.5. ENABLE FACE DETECTION

This command enables (or disables) face detection, facial expression detection, blink and gaze detection. Disabling one or more of these features reduces the number of events sent by Vector, and reduces his processing overhead.

post: "/v1/enable_face_detection"

### 45.5.1 Request

The EnableFaceDetectionRequest structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *enable* | bool | If true, face detection (and recognition) is enabled; otherwise face detection processes are disabled. |
| *enable_blink_detection* | bool | If true, Vector will attempt "to detect how much detected faces are blinking." Note: the blink amount is not reported. |
| *enable_expression_estimation* | bool | If true, Vector will attempt to estimate facial expressions. |
| *enable_gaze_detection* | bool | If true, Vector will attempt "to detect where detected faces are looking." Note: the gaze direction is not reported. |
| *enable_smile_detection* | bool | If true, Vector will attempt "to detect smiles in detected faces." Note: the smile is not reported. |

***Table 89:*** *EnableFaceDetectionRequest JSON structure*

### 45.5.2 Response

The EnableFaceDetectionResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 90:*** *EnableFaceDetectionResponse JSON structure*

## 45.6. ERASE ALL ENROLLED FACES

This command is used to erase all of the known faces (and their identity).

post: "/v1/erase_all_enrolled_faces"

### 45.6.1 Request

The EraseAllEnrolledFacesRequest structure has no fields.

### 45.6.2 Response

The EraseAllEnrolledFacesResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 91:*** *EraseAllEnrolledFacesResponse JSON structure*

## 45.7. ERASE ENROLLED FACE BY ID

This command is used to erase the indentify feature (and identity) of a known face.

post: "/v1/erase_enrolled_face_by_id"

### 45.7.1 Request

The EraseEnrolledFaceByIDRequest structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *face_id* | int32 | The identifier code for the face to erase. |

***Table 92:*** *EraseEnrolledFaceByIDRequest JSON structure*

### 45.7.2 Response

The EraseEnrolledFaceByIDResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 93:*** *EraseEnrolledFaceByIDResponse JSON structure*

## 45.8. FIND FACES

This causes Vector to look around for faces. This is carried out by the TBD behaviour.

post: "/v1/find_faces"

### 45.8.1 Request

The FindFacesRequest structure has no fields.

### 45.8.2 Response

The FindFacesResponse structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *result* | BehaviorResults | |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 94:*** *FindFacesResponse JSON structure*

## 45.9. REQUEST ENROLLED NAMES

This command is used to list the faces known to Vector, their names, and some other useful information.

post: "/v1/request_enrolled_names"

### 45.9.1 Request

The RequestEnrolledNamesRequest structure has no fields.

### 45.9.2 Response

The RequestEnrolledNamesRequest structure has the following fields:

***Table 95:*** *RequestEnrolledNames Response JSON structure*

| Field | Type | Description |
|---|---|---|
| *faces* | LoadedKnownFace[] | An array of the faces that are associated with names. |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

The LoadedKnownFace structure has the following fields:

***Table 96:*** *LoadedKnownFace JSON structure*

| Field | Type | Units | Description |
|---|---|---|---|
| *face_id* | int32 | | The identifier code for the face. |
| *last_seen_seconds_since_epoch* | int64 | *seconds* | The timestamp of the time the face was last seen |
| *name* | name | | The name associated with the face. |
| *seconds_since_first_enrolled* | int64 | *seconds* | The number of seconds since the face was first associated with a name and entered into the known faces database. |
| *seconds_since_last_seen* | int64 | *seconds* | The number of seconds since the face was last seen |
| *seconds_since_last_updated* | int64 | *seconds* | The number of seconds since (?) the name associated with the face was last changed.(?) |

## 45.10. SET FACE TO ENROLL

This command is can used to assign a name to unrecognized face, or to update the recognition pattern (and name) for an already known face. This command initiates a behaviour that can be configured.

post: "/v1/set_face_to_enroll"

### 45.10.1 Request

The SetFaceToEntrollRequest structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *name* | string | The name to associate with the face. |
| *observed_id* | int32 | If non-zero, the identifier code for a specific observed face to enroll. Note the identifier is negative if the face is not already recognized, positive if it has been recognized. If zero, Vector will use the next face he sees. |
| *save_id* | int32 | If non-zero, Vector will use this ID as the ID for the face. (Note: this must be "the ID of an existing face"). If zero, Vector will use the observedID for the ID. |
| *save_to_robot* | bool | If true, "save to robot's NVStorage when done (NOTE: will (re)save everyone enrolled!)" |
| *say_name* | bool | If true, "play say-name/celebration animations on success before completing." |
| *use_music* | bool | If true, "starts special music during say-name animations (will leave music playing!)" |

***Table 97:*** *SetFaceToEntrollRequest JSON structure*

### 45.10.2 Response

The SetFaceToEntrollResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 98:*** *SetFaceToEntrollResponse JSON structure*

## 45.11. UPDATE ENROLLED FACE BY ID

This command is used to change the name associated with a face.

post: "/v1/update_enrolled_face_by_id"

### 45.11.1 Request

The UpdateEnrolledFaceByIDRequest structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *face_id* | int32 | The identifier code for the face. |
| *new_name* | string | The new name to associate with the face. |
| *old_name* | string | The name associated (until now) with the face. This name must match the one Vector has for the face_id. If not the command will not be honored. |

***Table 99:*** *UpdateEnrolledFaceByIDRequest JSON structure*

### 45.11.2 Response

The UpdateEnrolledFaceByIDResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 100:*** *UpdateEnrolledFaceByIDResponse JSON structure*

# 46. FEATURES & ENTITLEMENTS

Vector has granular features that can be enabled and disabled thru the use of feature flags. This section describes the queries related to list Vector's features flags, and their state. For a description of feature flags, see Chapter 18 *Settings, Preferences, Features, and Statistics.* For a list of the features, and a description of each, see Appendix H.

Note: the API does not include the ability to enable a feature.

## 46.1. ENUMERATIONS

### 46.1.1 UserEntitlement

The UserEntitlement enumeration has the following named values:

| Name | Value | Description |
|---|---|---|
| *KICKSTARTER_EYES* | 0 | |

***Table 101:*** *UserEntitlement Enumeration*

## 46.2. GET FEATURE FLAG

The request the current setting of a feature flag.

post: "/v1/feature_flag"

### 46.2.1 Request

The FeatureFlagRequest message has the following fields:

| Field | Type | Description |
|---|---|---|
| *feature_name* | string | |

***Table 102:*** *FeatureFlagRequest JSON structure*

### 46.2.2 Response

The FeatureFlagResponse type has the following fields:

| Field | Type | Description |
|---|---|---|
| *feature_enabled* | bool | |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |
| *valid_feature* | bool | |

***Table 103:*** *FeatureFlagResponse JSON structure*

## 46.3. GET FEATURE FLAG LIST

To get a list of the current feature flags.

post: "/v1/feature_flag_list"

### 46.3.1 Request

The following is streamed… to the robot?

| Field | Type | Description |
|---|---|---|
| *request_list* | string | |

*Table 104: FeatureFlagListRequest JSON structure*

### 46.3.2 Response

The FeatureFlagListResponse type has the following fields:

| Field | Type | Description |
|---|---|---|
| *list* | string[] | An array of the feature flags |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

*Table 105: FeatureFlagListResponse JSON structure*

## 46.4. UPDATE USER ENTITLEMENTS

UpdateUserEntitlements

Post: "/v1/update_user_entitlements"

### 46.4.1 Request

The UpdateUserEntitlementsRequest has the following fields:

| Field | Type | Description |
|---|---|---|
| *user_entitlements* | UserEntitlementsConfig | |

***Table 106:*** *JSON Parameters for UpdateUserEntitlementsRequest*

The UserEntitlementsConfig has the following fields:

| Field | Type | Description |
|---|---|---|
| *kickstarter_eyes* | bool | |

***Table 107:*** *JSON Parameters for UserEntitlementsConfig*

### 46.4.2 Response

The UpdateUserEntitlementsResponse type has the following fields:

| Field | Type | Description |
|---|---|---|
| *code* | ResultCode | |
| *doc* | Jdoc | |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or |

*Table 108: UpdateUserEntitlementsResponse JSON structure*

| | | |
|---|---|---|
| | | an error code indicating why it was unable to be carried out. |

# 47. JDOCS

This section discussed the commands for "Jdocs" (short for "JSON Documents"), which are JSON objects that are passed to Vic-Engine and Vic-Cloud. See the next chapter for interactions with a remote Jdocs server, using a sibling protocol.

## 47.1. ENUMERATIONS

### 47.1.1 JdocType

The JdocType enumeration has the following named values:

**Table 109:** *JdocType Enumeration*

| Name | Value | Description |
|---|---|---|
| *ACCOUNT_SETTINGS* | 2 | Refers to the owner's account settings |
| *ROBOT_LIFETIME_STATS* | 1 | Refers to the robot's settings (owner preferences) |
| *ROBOT_SETTINGS* | 0 | Refers to the robot's lifetime stats. |
| *USER_ENTITLEMENTS* | 3 | Refers to the owner's entitlements. |

Items of these types are described in more detail in Chapter 18.

### 47.1.2 ResultCode

The ResultCode enumeration has the following named values:

**Table 110:** *ResultCode Enumeration*

| Name | Value | Description |
|---|---|---|
| *ERROR_UPDATE_IN_PROGRESS* | 1 | |
| *SETTINGS_ACCEPTED* | 0 | |

## 47.2. STRUCTURES

### 47.2.1 JDoc

The Jdoc type has the following fields:

**Table 111:** *JSON Jdoc structure*

| Field | Type | Description |
|---|---|---|
| *client_meta* | string | |
| *doc_version* | uint64 | |
| *fmt_version* | uint64 | |
| *json_doc* | string | |

### 47.2.2 NamedJDoc

The NamedJdoc type has the following fields:

| Field | Type | Description |
|---|---|---|
| *doc* | Jdoc | The JSON structure and meta-data about the document |
| *jdoc_type* | JdocType | The type of document provided in "doc" |

***Table 112:** JSON NamedJdoc structure*

## 47.3. EVENTS

### 47.3.1 JdocsChanged

The JdocsChanged message has the following fields:

| Field | Type | Description |
|---|---|---|
| *jdoc_types* | JdocType[] | |

***Table 113:** JSON JdocsChanged request structure*

## 47.4. PULL JDOCS

Post: "/v1/pull_jdocs"

### 47.4.1 Request

The PullJdocsRequest has the following fields:

| Field | Type | Description |
|---|---|---|
| *jdoc_types* | JdocType[] | |

***Table 114:** JSON PullJdocsRequest structure*

### 47.4.2 Response

The PullJdocResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *named_jdocs* | NamedJdoc[] | |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 115:** JSON PullJdocsResponse structure*

# 48. MISC, ACCESSORIES AND CUSTOM OBJECTS

## 48.1. DELETECUSTOMOBJECTS

DeleteCustomObjects

Post: "/v1/delete_custom_objects"

### 48.1.1 Request

### 48.1.2 Response

## 48.2. UPLOADDEBUGLOGS

TBD: Request that the logs be uploaded to the server for analysis.

Post: "/v1/upload_debug_logs"

### 48.2.1 Request

The UploadDebugLogsRequest structure has no fields.

### 48.2.2 Response

The UploadDebugLogsResponse structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |
| *url* | string | |

***Table 116:*** *UploadDebugLogsResponse JSON structure*

# 49. MOTION CONTROL

## 49.1. DRIVE STRAIGHT

Tells Vector to drive in a straight line.

Note: "Vector will drive for the specified distance (forwards or backwards) Vector must be off of the charger for this movement action. Note that actions that use the wheels cannot be performed at the same time, otherwise you may see a TRACKS_LOCKED error."

### 49.1.1 Request

The DriveStraightRequest has the following fields:

*Table 117: DriveStraightRequest JSON structure*

| Field | Type | Units | Description |
|---|---|---|---|
| *dist_mm* | float | *mm* | The distance to drive. (Negative is backwards) |
| *id_tag* | int32 | | |
| *is_absolute* | uint32 | | If 0, turn by angle_rad relative to the current orientation. If 1, turn to the absolute angle given by angle_rad. |
| *num_retries* | int32 | | Maximum of times to attempt to move the head to the height. A retry is attempted if Vector is unable to reach the target angle |
| *should_play_animation* | bool | | If true, "play idle animations whilst driving (tilt head, hum, animated eyes, etc.)" |
| *speed_mmps* | float | *mm/sec* | The speed to drive at. This should be positive. |

### 49.1.2 Response

The DriveStraightResponse has the following fields:

*Table 118: DriveStraightResponse JSON structure*

| Field | Type | Description |
|---|---|---|
| *response* | ActionResult | Whether the action is able to be run. If not, an error code indicating why not. |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

## 49.2. DRIVE WHEELS

Sets the speed and acceleration for Vector's wheel motors.

### 49.2.1 Request

The DriveWheelsRequest has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *left_wheel_mmps* | float | *mm/sec* | The initial speed to set the left wheel to. |
| *left_wheel_mmps2* | float | *mm/sec*$^2$ | How fast to increase the speed of the left wheel. |
| *right_wheel_mmps* | float | *mm/sec* | The initial speed to set the right wheel to. |
| *right_wheel_mmps2* | float | *mm/sec*$^2$ | How fast to increase the speed of the right wheel. |

***Table 119:** DriveWheelsRequest JSON structure*

To unlock the wheels, set all values to 0.

### 49.2.2 Response

The DriveWheelsResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 120:** DriveWheelsResponse JSON structure*

## 49.3. MOVE HEAD

Move Vector's head

### 49.3.1 Request

The MoveHeadRequest has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *speed_rad_per_sec* | float | *rad/sec* | The speed to drive the head motor at. Positive values are up, negative move down. A value of 0 will unlock the head track. |

***Table 121:** MoveHeadRequest JSON structure*

### 49.3.2 Response

The MoveHeadResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 122:** MoveHeadResponse JSON structure*

## 49.4. MOVE LIFT

Move Vector's lift

### 49.4.1 Request

The MoveLiftRequest has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *speed_rad_per_sec* | float | *rad/sec* | The speed to drive the lift at. Positive values are up, negative move down. A value of 0 will unlock the lift track. |

***Table 123:*** *MoveLiftRequest JSON structure*

### 49.4.2 Response

The MoveLiftResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 124:*** *MoveLiftResponse JSON structure*

## 49.5. SET HEAD ANGLE

Tell Vector's head to move to a given angle.

### 49.5.1 Request

The SetHeadAngleRequest has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *accel_rad_per_sec2* | float | *rad/sec²* | How fast to increase the speed the head is moving at |
| *angle_rad* | float | *rad* | The target angle to move Vector's head to. This should be in the range -22.0° to 45.0°. |
| *duration_sec* | float | *sec* | "Time for Vector's head to move in seconds. A value of zero will make Vector try to do it as quickly as possible." |
| *id_tag* | int32 | | |
| *max_speed_rad_per_sec* | float | *rad/sec* | The maximum speed to move the head at. (This clamps the speed from further acceleration.) |
| *num_retries* | int32 | *count* | Maximum of times to attempt to move the head to the height. A retry is attempted if Vector is unable to reach the target angle |

*Table 125: SetHeadAngleRequest JSON structure*

### 49.5.2 Response

The SetHeadAngleResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *response* | ActionResult | Whether the action is able to be run. If not, an error code indicating why not. |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

*Table 126: SetHeadAngleResponse JSON structure*

## 49.6. SET LIFT HEIGHT

Tell Vector's lift to move to a given height.

### 49.6.1 Request

The SetLiftRequest has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *accel_rad_per_sec2* | float | *rad/sec²* | How fast to increase the speed the lift is moving at |
| *duration_sec* | float | *sec* | "Time for Vector's lift to move in seconds. A value of zero will make Vector try to do it as quickly as possible." |
| *height_mm* | float | *mm* | The target height to raise the lift to.<br>Note: the python API employs a different range for this parameter |
| *id_tag* | int32 | | |
| *max_speed_rad_per_sec* | float | *rad/sec* | The maximum speed to move the lift at. (This clamps the speed from further acceleration.) |
| *num_retries* | int32 | *count* | Maximum of times to attempt to move the lift to the height. A retry is attempted if Vector is unable to reach the target height |

***Table 127:*** *SetLiftRequest JSON structure*

### 49.6.2 Response

The SetLiftResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *response* | ActionResult | Whether the action is able to be run. If not, an error code indicating why not. |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 128:*** *SetLiftResponse JSON structure*

## 49.7. STOP ALL MOTORS

Stop all motor commands for the head, lift and wheels

### 49.7.1 Request

The StopAllMotorsRequest structure has no fields.

### 49.7.2 Response

The StopAllMotorsResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 129:*** *StopAllMotorsResponse JSON structure*

## 49.8. TURN IN PLACE

Turn the robot around its current position.

Note: "Vector must be off of the charger for this movement action. Note that actions that use the wheels cannot be performed at the same time, otherwise you may see a TRACKS_LOCKED error."

### 49.8.1 Request

The TurnInPlaceRequest has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *accel_rad_per_sec2* | float | *rad/sec²* | How fast to increase the speed the body is moving at |
| *angle_rad* | float | *rad* | If isAbsolute is 0, turn relative to the current heading by this number of radians; positive means turn left, negative is turn right.. Otherwise, turn to the absolute orientation given by this angle. |
| *id_tag* | int32 | | |
| *is_absolute* | uint32 | | If 0, turn by angle_rad relative to the current orientation. If 1, turn to the absolute angle given by angle_rad. |
| *num_retries* | int32 | | Maximum of times to attempt to move the head to the height. A retry is attempted if Vector is unable to reach the target angle |
| *speed_rad_per_sec* | float | *rad/sec* | The speed to move around the arc. |
| *tol_rad* | float | *count* | "The angular tolerance to consider the action complete (this is clamped to a minimum of 2 degrees internally)." |

***Table 130:*** *TurnInPlaceRequest JSON structure*

### 49.8.2 Response

The TurnInPlaceResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *response* | ActionResult | Whether the action is able to be run. If not, an error code indicating why not. |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 131:*** *TurnInPlaceResponse JSON structure*

# 50. MOTION SENSING AND ROBOT STATE

## 50.1. ENUMERATIONS

### 50.1.1 RobotStatus

The RobotStatus is a bit mask used to indicate the status of Vector. It is used in the RobotState message. The enumeration has the following named bits (any number may be set). Note that some bits have two names; the second name is one employed by Anki's python SDK.

| Name | Value | Description |
|---|---|---|
| *ROBOT_STATUS_NONE* | $00000_{16}$ | |
| *ROBOT_STATUS_IS_MOVING*<br>*ROBOT_STATUS_ARE_MOTORS_MOVING* | $00001_{16}$ | This bit is set "if Vector is currently moving any of his motors (head, arm or wheels/treads)." |
| *ROBOT_STATUS_IS_CARRYING_BLOCK* | $00002_{16}$ | This bit is set "if Vector is currently carrying a block." |
| *ROBOT_STATUS_IS_PICKING_OR_PLACING*<br>*ROBOT_STATUS_IS_DOCKING_TO_MARKER* | $00004_{16}$ | This bit is set "if Vector has seen a marker and is actively heading toward it (for example his charger or cube)." |
| *ROBOT_STATUS_IS_PICKED_UP* | $00008_{16}$ | This bit is set "if Vector is currently picked up (in the air)," being held or is on his side. Vector "uses the IMU data to determine if the robot is not on a stable surface with his treads down." If Vector is not on stable surface (with his treads down), this bit is set. |
| *ROBOT_STATUS_IS_BUTTON_PRESSED* | $00010_{16}$ | This bit is set "if Vector's button is pressed." |
| *ROBOT_STATUS_IS_FALLING* | $00020_{16}$ | This bit is set "if Vector is currently falling." |
| *ROBOT_STATUS_IS_ANIMATING* | $00040_{16}$ | This bit is set "if Vector is currently playing an animation." |
| *ROBOT_STATUS_IS_PATHING* | $00080_{16}$ | This bit is set "if Vector is currently traversing a path." |
| *ROBOT_STATUS_LIFT_IN_POS* | $00100_{16}$ | This bit is set "if Vector's arm is in the desired position." It is clear "if still trying to move it there." |
| *ROBOT_STATUS_HEAD_IN_POS* | $00200_{16}$ | This bit is set "if Vector's head is in the desired position." It is clear "if still trying to move there." |
| *ROBOT_STATUS_CALM_POWER_MODE* | $00400_{16}$ | This bit is set "if Vector is in calm power mode. Calm power mode is generally when Vector is sleeping or charging." |

***Table 132:*** *RobotStatus Enumeration*

| | | |
|---|---|---|
| *reserved* | $00800_{16}$ | *This bit is not defined* |
| *ROBOT_STATUS_IS_ON_CHARGER* | $01000_{16}$ | This bit is set "if Vector is currently on the charger." Note: Vector may be on the charger without charging. |
| *ROBOT_STATUS_IS_CHARGING* | $02000_{16}$ | This bit is set "if Vector is currently charging." |
| *ROBOT_STATUS_CLIFF_DETECTED* | $04000_{16}$ | This bit is set "if Vector detected a cliff using any of his four cliff sensors." |
| *ROBOT_STATUS_ARE_WHEELS_MOVING* | $08000_{16}$ | This bit is set "if Vector's wheels/treads are currently moving." |
| *ROBOT_STATUS_IS_BEING_HELD* | $10000_{16}$ | This bit is set "if Vector is being held."<br>Note: ROBOT_STATUS_IS_PICKED_UP will also be set when this bit is set.<br>Vector "uses the IMU to look for tiny motions that suggest the robot is actively being held in someone's hand." This is used to distinguish from other cases, such as falling, on its side, etc. |
| *ROBOT_STATUS_IS_MOTION_DETECTED*<br>*ROBOT_STATUS_IS_ROBOT_MOVING* | $20000_{16}$ | This bit is set "if Vector is in motion. This includes any of his motors (head, arm, wheels/tracks) and if he is being lifted, carried, or falling." |

Note: all quotes above are from the python SDK.

## 50.2. STRUCTURES

### 50.2.1 AccelData

This is used to report the accelerometer readings, as part of the RobotState structure. The accelerometer is located in Vector's head, so its XYZ axes are not the same as Vector's body axes. When motionless, the accelerometer can be used to calculate the angle of Vectors head. The AccelData has the following fields:

**Table 133:** *AccelData JSON structure*

| Field | Type | Units | Description |
|---|---|---|---|
| *x* | float | *mm/s²* | The acceleration along the accelerometers X axis. |
| *y* | float | *mm/s²* | The acceleration along the accelerometers Y axis. |
| *z* | float | *mm/s²* | The acceleration along the accelerometers Z axis. |

When at rest, there will be a constant 9810 $mm/s^2$ downward acceleration from gravity. This most likely will be distributed across multiple axes.

### 50.2.2 GyroData

This is used to report the gyroscope readings, as part of the RobotState structure. The gryoscope is located in Vector's head, so its XYZ axes are not the same as Vector's body axes. The GryroData has the following fields:

**Table 134:** *GyroData JSON structure*

| Field | Type | Units | Description |
|---|---|---|---|
| *x* | float | *radian/s* | The angular velocity around the X axis. |
| *y* | float | *radian/s* | The angular velocity around the Y axis. |

| | | | |
|---|---|---|---|
| *z* | float | *radian/s* | The angular velocity around the Z axis. |

### 50.2.3 RobotState

The RobotState structure is periodically by the EventStream command. The structure has the following fields:

***Table 135:** RobotState JSON structure*

| Field | Type | Units | Description |
|---|---|---|---|
| *accel* | AccelData | | The accelerometer readings |
| *carrying_object_id* | int32 | | The identifier of the cube (or other object) being carried. If no object is being carried, this will be -1. |
| *carrying_object_on_top_id* | int32 | | *Not supported* |
| *gyro* | GyroData | | The gyroscope readings |
| *head_angle_rad* | float | *radian* | The angle of Vector's head (how much it is tilted up or down). |
| *head_tracking_object_id* | int32 | | The identifier "of the object the head is tracking to." If no object is being tracked, this will be -1. |
| *last_image_time_stamp* | uint32 | | "The robot's timestamp for the last image seen." |
| *left_wheel_speed_mmps* | float | *mm/s* | The speed of Vector's left wheel. |
| *lift_height_mm* | float | *mm* | "Height of Vector's lift from the ground." |
| *localized_to_object_id* | int32 | | The identifier "of the object that the robot is localized to." If no object, this will be -1. |
| *pose* | PoseStruct | | "The current pose (position and orientation) of Vector." |
| *pose_angle_rad* | float | *radian* | "Vector's pose angle (heading in X-Y plane)." |
| *pose_pitch_rad* | float | *radian* | "Vector's pose pitch (angle up/down)." |
| *right_wheel_speed_mmps* | float | *mm/s* | The speed of Vector's right wheel. |
| *prox_data* | ProxData | | The time-of-flight proximity sensor readings. |
| *status* | uint32 | | A bit map of active states of Vector; the bits are described in the RobotStatus enumeration.<br>"This status provides a simple mechanism to, for example, detect if any of Vector's motors are moving, determine if Vector is being held, or if he is on the charger." |
| *touch_data* | TouchData | | The touch sensor readings. |

Note: all quotes above are from the python SDK.

### 50.3. ENABLE MOTION DETECTION

Enables sending RobotObservedMotion events

Post: "/v1/enable_motion_detection"

#### 50.3.1 Request

#### 50.3.2 Response

The EnableMotionDetectiontResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 136:*** *EnableMotionDetectiont Response JSON structure*

## 51. SETTINGS AND PREFERENCES

This section describes the commands and queries related to settings and preferences on Vector. For a description of the settings and what they mean, see Chapter 18 *Settings, Preferences, Features, and Statistics.* That chapter includes definitions for the following types:

- AccountSettingsConfig
- RobotSettingsConfig

### 51.1. STRUCTURES

#### 51.1.1 AccountSettingsConfig

The AccountSettingsConfig type has the following fields:

| Field | Type | Description |
|---|---|---|
| *app_locale* | string | |
| *data_collection* | boolean | |

***Table 137:*** *AccountSetting JSON structure*

### 51.2. UPDATESETTINGS

Post: "/v1/update_settings"

#### 51.2.1 Request

The UpdateSettingsRequest has the following fields:

| Field | Type | Description |
|---|---|---|
| *settings* | RobotSettingsConfig | |

***Table 138:*** *UpdateSettingsRequest JSON structure*

### 51.2.2 Response

The UpdateSettingsResponse type has the following fields:

| Field | Type | Description |
|---|---|---|
| *code* | ResultCode | |
| *doc* | Jdoc | |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 139:*** *UpdateSettingsResponse JSON structure*

## 51.3. UPDATEACCOUNTSETTINGS

Post: "/v1/update_account_settings"

### 51.3.1 Request

The UpdateAccountsSettingsRequest has the following fields:

| Field | Type | Description |
|---|---|---|
| *account_settings* | AccountSettingsConfig | |

***Table 140:*** *JSON Parameters for UpdateAccountSettings Request*

### 51.3.2 Response

The UpdateAccountsSettingsResponse type has the following fields:

| Field | Type | Description |
|---|---|---|
| *code* | ResultCode | |
| *doc* | Jdoc | |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 141:*** *UpdateAccountSettings Response JSON structure*

# 52. SOFTWARE UPDATES

These commands are siblings to the OTA Update and related commands in Chapter 12 Bluetooth LE protocol. However, they differ: in some cases, less information, in others present the same information in different ways.

## 52.1. ENUMERATIONS

### 52.1.1 UpdateStatus

The UpdateStatus enumeration has the following named values:

| Name | Value | Description |
|---|---|---|
| *IN_PROGRESS_DOWNLOAD* | 2 | |
| *NO_UPDATE* | 0 | |
| *READY_TO_INSTALL* | 1 | |

***Table 142:*** *UpdateStatus Enumeration*

## 52.2. START UPDATE ENGINE

"StartUpdateEngine cycles the update-engine service (to start a new check for an update) and sets up a stream of UpdateStatusResponse events."

Post: "/v1/start_update_engine"

This command uses the same request and response structures as CheckUpdateStatus

## 52.3. CHECK UPDATE STATUS

"CheckUpdateStatus tells if the robot is ready to reboot and update."

Post: "/v1/check_update_status"

### 52.3.1 Request

The CheckUpdateStatusRequest structure has no fields.

### 52.3.2 Response

This is streamed set of update status. The CheckUpdateStatusResponse type has the following fields:

| Field | Type | Description |
|---|---|---|
| *expected* | int64 | The number of bytes expected to be downloaded |
| *progress* | int64 | The number of bytes downloaded |
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |
| *update_status* | UpdateStatus | |
| *update_version* | string | |

***Table 143:*** *CheckUpdateStatusResponse JSON structure*

## 52.4. UPDATE AND RESTART

Post: "/v1/update_and_restart"

### 52.4.1 Request

The UpdateAndRestartRequest structure has no fields.

### 52.4.2 Response

The UpdateAndRestartResponse has the following fields:

| Field | Type | Description |
|---|---|---|
| *status* | ResponseStatus | A generic status of whether the request was able to be carried out, or an error code indicating why it was unable to be carried out. |

***Table 144:*** *UpdateAndRestartResponse JSON structure*

CHAPTER 14

# The Cloud Services

This chapter describes the remote servers that provide functionality for Vector.

- JSON document storage server
- The crash uploader
- The diagnostic logger
- The token/certificate system
- The natural language processing

## 53. CONFIGURATION

The server URLs are specified in "/anki/data/assets/cozmo_resources/config/server_config.json" (The path to this JSON file is hardcoded in the vic-cloud binary.)

**Table 145:** *The cloud services configuration file*

| Element | Description & Notes |
|---|---|
| *appkey* | A base64 token used to communicate with servers. "oDoa0quieSeir6goowai7f" |
| *check* | The server to use for connection checks |
| *chipper* | The natural language processing server |
| *jdocs* | The remote JSON storage server |
| *logfiles* | The server to upload log files to |
| *tms* | The token server where Vector gets authentication items like certificates and tokens |

The crash upload URL is given in /anki/etc/vic-crashuploader.env

The OTA download URL is given in /anki/etc/update-engine.env

The DAS server to contact is given in /anki/data/assets/cozmo_resources/config/DASConfig.json (This path is hardcoded in vic-DASMgr)

## 54. JDOCS SERVER

The Vic-Cloud services stores information on a "JDocs" server. This unusual name appears to be short for "JSON Documents."

Vic-Cloud uses the "jdocs" tag in the cloud services configuration file to know which server to contact. It uses the file

/anki/data/assets/cozmo_resources/config/engine/jdocs_config.json to adjust how often it contacts the server. (The path to this JSON file is hardcoded in libcozmo_engine.)

The interactions are basic: store, read, and delete a JSON blob by an identifier. The description below[36] gives the JSON keys, value format. It is implemented as gRPC/protobuf interaction over HTTP.

## 54.1. JDOCS INTERACTION

The JDoc message has the following fields

| Field | Type | Description |
|---|---|---|
| *client_meta* | string | |
| *doc_version* | uin64 | |
| *fmt_version* | uint64 | |
| *json_doc* | string | |

***Table 146:*** *JSON Parameters for JDoc request*

## 54.2. DELETE DOCUMENT

### 54.2.1 Request

DeleteDocReq

- account
- thing – the thing id is a 'vic:' followed by the serial number
- doc_name

### 54.2.2 Response

DeleteDocResp

## 54.3. ECHO TEST

### 54.3.1 Request

EchoReq

- data

### 54.3.2 Response

EchoResp

- data

[36] The protocol was specified in Google Protobuf. Vic-Cloud and Vic-Gateway were both written in Go. There is enough information in those binaries to reconstruct significant portions of the Protobuf specification in the future.

## 54.4. READ DOCUMENTS

### 54.4.1 Request

ReadDocsReq

- account
- thing
- items

### 54.4.2 Response

ReadDocsResp

- items

## 54.5. READ DOCUMENT ITEM

### 54.5.1 Request

ReadDocsReq_Item

- doc_name
- my_doc_version

### 54.5.2 Response

ReadDocsResp_Item

- status (int)
- doc

## 54.6. WRITE DOCUMENT

### 54.6.1 Request

WriteDocReq

- account
- thing
- doc_name
- doc

### 54.6.2 Response

WriteDocResp

- status (int)
- latest_doc_version

# 55. NATURAL LANGUAGE PROCESSING

The "knowledge graph" Q&A server is done by Sound Hound.

## 56. LOG UPLOADER

The logs are uploading by performing a HTTP PUT to the server. The URL is the "logfiles" URL in the server configuration file, with the compressed log file "/victor-${esn}-${timestamp}-${pid}.log.gz" appended. The HTTP headers are:

**Table 147:** *Log upload HTTP header fields*

| HTTP header | Description |
|---|---|
| *Anki-App-Key:* | The appKey from the server configuration file. |
| *Usr-RobotESN:* | Vector's serial number |
| *Usr-RobotOSRevision:* | The OS revision string from /etc/os-version-rev |
| *Usr-RobotOSVersion:* | The OS version string from /etc/os-version |
| *Usr-RobotRevision:* | The Anki revision string from /anki/etc/revision |
| *Usr-RobotTimestamp:* | The time of Vector's internal clock. |
| *Usr-RobotVersion:* | The Anki version string from /anki/etc/version |
| *Usr-Username:* | |

Note: the log uploader does not appear to be enabled in the production software.

## 57. CRASH UPLOADER

Minidumps are uploaded to a backtrace.io server. The URL (including the key) is hard coded in anki-crashuploader. This is done using a POST. The HTTP headers are:

**Table 148:** *Crash upload form fields*

| Form fields | Description |
|---|---|
| *attachment_messages.log* | The ".log" file associated with the minidump. This is optional; only included if /run/das_allow_upload exists |
| *hostname* | ${hostname} |
| *robot.esn* | Vector's serial number |
| *robot.os_version* | The OS version string from /etc/os-version |
| *robot.anki_version* | The Anki version string from /anki/etc/version |
| *upload_file* | The minidump ".dmp" file |

## 58. DAS MANAGER

DAS Manager uploads TBD to an Amazon "Simple Queue Service" (SQS) server. Amazon's API uses the following key/value pairs in a URL encoded form:

**Table 149:** *DAS Manager SQS key-value pairs*

| Keys | Value |
|---|---|
| *Action* | SendMessage |
| *MessageAttribute.1.Name* | DAS-Transport-Version |
| *MessageAttribute.1.Value.DataType* | Number |
| *MessageAttribute.1.Value.StringValue* | 2 |

| | |
|---|---|
| *MessageAttribute.2.Name* | Content-Encoding |
| *MessageAttribute.2.Value.DataType* | String |
| *MessageAttribute.2.Value.StringValue* | gzip, base64 |
| *MessageAttribute.3.Name* | Content-Type |
| *MessageAttribute.3.Value.DataType* | String |
| *MessageAttribute.3.Value.StringValue* | application/vnd.anki.json; format=normal; product=vic |
| *MessageBody* | |
| *Version* | 2012-11-05[37] |

Note: there may be a body of compressed JSON data. These values are hardcoded in vic-dasmgr and libcozmo_engine

The literature for their Elemental toolkit

> Unless you create an account and log in, Analytics Data is stored under a unique ID and not connected to you.

## 59. REFERENCES AND RESOURCES

Davis, Jason; *File Attachments in Backtrace*, Backtrace.io
https://help.backtrace.io/en/articles/1852523-file-attachments-in-backtrace

[37] This date is very far in the past, before Vector or Cozmo were developed. This was the time frame of the Overdrive product development.

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

PART IV

# Advanced Functions

This part describes items that are Vector's primary function.

- IMAGE PROCESSING. A look at how Vector vision system, photographs, and faces works
- MAPPING, NAVIGATION. A look at Vector's mapping and navigation systems
- ACCESSORIES, CHARGING DOCK, COMPANION CUBE.

*drawing by Steph Dere*

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

CHAPTER 15

# Image Processing

This chapter describes the image processing system:

- Camera operation including calibration
- Visual stimulation
- Recognizing symbols and specially marked objects
- Detecting faces, recognizing people, and estimating emotion
- Hand, pet and other object detection
- Taking photos
- Sending video stream to the SDK
- Vision is primarily used for navigation purposes: Recognizing the floor (or ground), odometery and "simultaneous localization and mapping"

## 60. CAMERA OPERATION

Vector has a 1280x720 camera with a wide field of view to see around it without moving its head, similar to how an animal can see a wide area around it by moving its eyes. The camera is connected to the processor through a MIPI interface. The drivers, separate service and the eventually passes to Vic-engine for the processing:

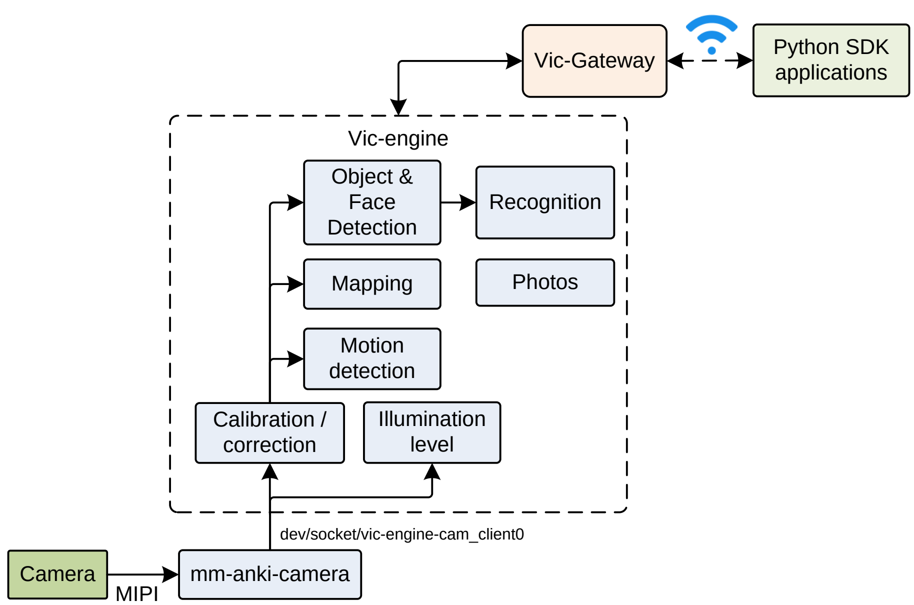



***Figure 41:*** *The camera architecture*

Vector visually recognizes some elements in its environment:

- Special visual markers; Vector treats all marked objects as moveable… and all other objects in its driving are as fixed & unmovable.
- Faces
- Hands
- Pets (feature not completed)
- Other objects, like fruit, etc. (feature not completed)
- LASER pointers (feature not completed)

## 60.1. CAMERA CALIBRATION

The camera is calibrated at manufacturing time. This is necessary so that the Vector can accurately dock with a cube, getting the small lift fingers into the cube's holes. The calibration primarily compensates for the image being slightly offset, and unit to unit variation of focal length.

Vector's camera has ~120° diagonal *field of view*.[38] For comparison the iPhone's camera has a 73˚ field of view, and the human eye is approximately 95˚. The cropped sensor image has a 90° horizontal field of view and a 50° vertical field of view.

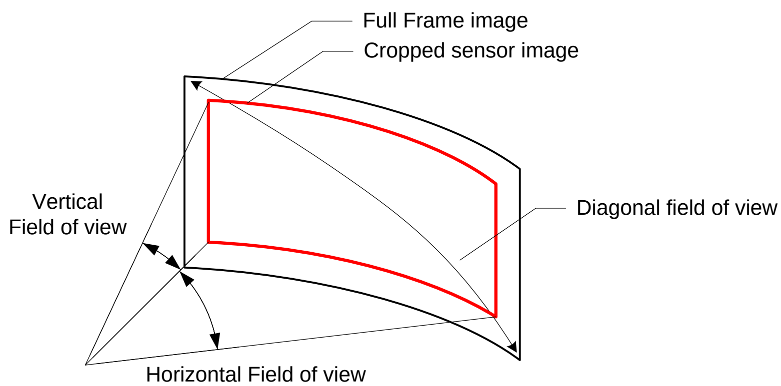



***Figure 42:** The camera field of view*

Vector's calibration uses focal length instead of field of view. The two values are related:

$$fieldOfView = 2\arctan\frac{sensor\ size}{2\ focalLength}$$

***Equation 1:** Relationship between field of view and focal length*

[38] The press for Vector reported a 120° field of view, but should be discounted as this number does not match the frame field of view numbers given in the SDK documentation.

The following structure is reported in the robot logs for the camera calibration:

| Field | Type | Description |
|---|---|---|
| *cx*<br>*cy* | float | "The position of the optical center of projection within the image. It will be close to the center of the image, but adjusted based on the calibration of the lens at the factory." |
| *distortionCoeffs* | float[] | |
| *fx*<br>*fy* | float | The "focal length combined with pixel skew (as the pixels aren't perfectly square), so there are subtly different values for x and y." |
| *ncols* | int | The width of the image in pixels. The value given is 640. |
| *nrows* | int | The height of the image in pixels. The value given is 360 |
| *skew* | float | |

***Table 150:** The camera calibration JSON structure*

*Quotes are from Anki Cozmo SDK.*

"A full 3x3 calibration matrix for doing 3D reasoning based on the camera images would look like:"

$$\begin{pmatrix} focalLength_x & 0 & center_x \\ 0 & focalLength_y & center_y \\ 0 & 0 & 1 \end{pmatrix}$$

***Equation 2:** Camera calibration matrix*

Note: Since the width and height of the camera is half that of the 1280x720 value publically advertised, this suggests that the either the camera is calibrated (and typically used with) at half its full capacity; or the camera is really a 640x360 camera, and the image is scaled up to appear to be HD.

## 60.2. VISION MODES

The vision process happen at different rates, many execute together in a shared group.

The vision processing system has many detectors, and functions. Some have their software run at different rates. While most are independent of each other, they are often grouped together.

| Vision Mode | Executes with | Description and notes |
|---|---|---|
| *AutoExposure* | | |
| *BuildingOverheadMap* | | |
| *CompositingImages* | | |
| *ComputingStatistics* | | |
| *CroppedFaceDetection* | DetectingFaces | Used to detect faces that are obscured or partly out of view. |
| *CyclingExposure* | AutoExposure | |
| *DetectingBlinkAmount* | | |
| *DetectingBrightColors* | | |
| *DetectingFaces* | | Used for face detection, and to trigger facial identification. |

***Table 151:** The Vision processes*

| | | |
|---|---|---|
| *DetectingGaze* | | Detects the gaze and looks deep into their eyes with wonder and the hope of biscuits. |
| *DetectingHands* | | Used to detect hands (for purposes of pouncing on them). |
| *DetectingIllumination* | | |
| *DetectingLaserPoints* | | |
| *DetectingMarkers* | | Detects Vector's special square marker symbols. |
| *DetectingMotion* | | |
| *DetectingOverheadEdges* | | |
| *DetectingPeople* | | |
| *DetectingPets* | | |
| *DetectingSmileAmount* | | |
| *DisableMarkerDetection* | | |
| *EstimatingFacialExpression* | | |
| *FullFrameMarkerDetection* | DetectingMarkers | This part of the process of detecting Vector's special square marker symbols. |
| *FullHeightMarkerDetection* | DetectingMarkers | This part of the process of detecting Vector's special square marker symbols. |
| *ImageViz* | | |
| *FullWidthMarkerDetection* | DetectingMarkers | This part of the process of detecting Vector's special square marker symbols. |
| *MarkerDetectionWhileRotatingFast* | DetectingMarkers | This part of the process of detecting Vector's special square marker symbols. |
| *MeteringFromChargerOnly* | AutoExposure | |
| *MinGainAutoExposure* | AutoExposure | |
| *MirrorMode* | | Displays the camera image on the LCD |
| *SavingImages* | | |

Notes: has rate control on processing those

## 60.3. ILLUMINATION LEVEL SENSING

Vector estimations the amount of illumination in the room. Dark rooms would encourage him to go to sleep, while bright or changing illumination would encourage him to be active. The illumination is pretty each to compute: sum up the brightness of each pixel in the image, or the number of pixels above a threshold of brightness.

The camera is also used as an ambient light sensor when Vector is in low power mode (e.g. napping, or sleeping). In low power mode, the camera is suspended and not acquiring images. Although in a low power state, it is still powered. The software reads the camera's auto exposure/gain settings and uses these as an ambient light sensor. (This allows it to detect when there is activity and Vector should wake.)

## 61. THE CAMERA POSE: WHAT DIRECTION IS CAMERA POINTING IN?

The camera is located in Vector's head. The pose of Vector's camera – its position and orientation, including its tilt up or down, can be estimated from Vector's pose, the angle of his head, the known position of the camera within the head and the position of the joint around which the head swivels. Note: the values are for Cozmo, but are assumed to be representative of Vector:

```
# Neck joint relative to robot origin
NECK_JOINT_POSITION = [-13, 0, 49]

# camera relative to neck joint
HEAD_CAM_POSITION = [17.52, 0, -8]
DEFAULT_HEAD_CAM_POS = list(HEAD_CAM_POSITION)

DEFAULT_HEAD_CAM_ROTATION = [
  0,      -0.0698,   0.9976,
 -1,       0,        0,
  0,      -0.9976,  -0.0698 ]

# Compute pose from robot body to camera
# Start with a pose defined by the DEFAULT_HEAD_CAM_ROTATION (rotation matrix)
# and the initial position DEFAULT_HEAD_CAM_POS
default_head_pose = Matrix3d(DEFAULT_HEAD_CAM_ROTATION, DEFAULT_HEAD_CAM_POS)

# Rotate that by the head angle
rotation_vector = RotationVectorAroundYAxis(-robot.head_angle.radians);
current_head_pose = default_head_pose.rotate_by(rotation_vector)

# Get the neck pose (transform the initial offset by the robot's pose)
neck_pose = TransformPose(NECK_JOINT_POSITION, robot.pose)

# Precompose with robot-to-neck-pose
camera_pose = current_head_pose.pre_compose_with(neck_pose);
```

***Example 6:*** *Computing the camera pose source: Anki*[39]

[39] https://forums.anki.com/t/camera-matrix-for-3d-positionning/13254/5

## 62. MARKERS

Anki considered QR codes to mark accessories and special items… but they were universally rejected in the feedback received during development. So Anki created their own visual labeling system, starting with Cozmo. Vector has a newer set of visual labels that is not compatible with Cozmos. (There isn't a clear reason for the incompatibility.) The algorithm used is among the most documented of Anki's internally developed modules for Vector.

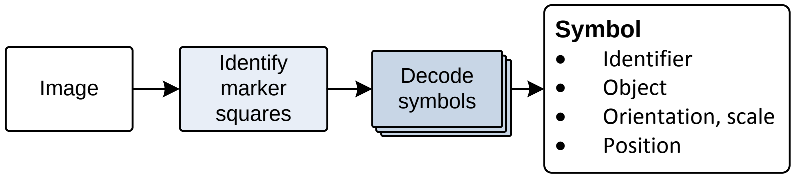



***Figure 43:*** *The processing of the image for symbols and objects*

A key characteristic of the markers is a big, bold square line around it:

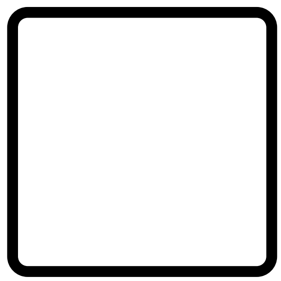


***Figure 44:*** *A typical rectangle around the visual markers*

The square is used to estimate the distance and relative orientation (pose) of the marker and the object is on. Vector, internally, knows the physical size of marker. The size of the square in the view — and being told how big the shape really is —lets Vector know enough to compute the likely physical distance to the marked item. And since the "true" mark has parallel lines, Vector can infer the pose (relative angles) of the surface the mark is on.

The process of finding and decoding the marker symbols is very straightforward, since there is quite a lot known about the structure of the marker image ahead of time. This allows the use of computation friendly algorithms.

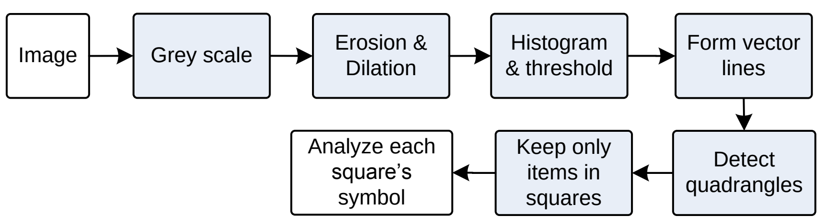



***Figure 45:*** *Preparing image for scanning for symbols*

The steps in processing are:

1. Acquire a gray scale image,
2. Apply classic erosion-dilation and Sobel transforms to build a vector representation (no pun intended) of the image; this is most familiar as "vector drawing" vs bitmap images
3. Detect the squares – the parallel and perpendicular lines – in the vector drawing. This will be the potential area that a symbol is in.
4. Analyze square to determine is size, and affine transform – how it is tilted up-and-down, and tilted away from the camera.

5. Screen the squares, tossing out that those that are horribly distorted,
6. Analyze the pixels in the square to identify the code

## 62.1. THE INITIAL PREPARATION STEPS

The image is initially prepared for analysis by:

1. The image is converted to grey scale, since color is of no value.
2. Performs (erosion, dilation) that strip out noise, fill in minor pixel gaps. There are no small features, so fine detail is not important.
3. The image is then converted to high-contrast black and white (there is no signal in grey scale). This is done by performing a histogram of the grey scale colors, finding a median value. This value is used as a threshold value: greys darker than this are consider black (a 1 bit), and all others are white (0 bit).

## 62.2. DETECT AND ANALYZE SQUARES

The detection of squares then:

1. Typically a pair of Sobel filters is applied to identify edges of the black areas, and the gradients (the x-y derivative) of the edges.
2. The adjacent (or nearby) pixels with similar gradients are connected together into a list. Straight line segments will have very consistent gradients along them. In other words, the bitmap is converted into a vector drawing. In jargon, this is called the *morphology.*
3. The lists of lines are organized into a containment tree. A bounding box (min and max positions of the points in the list) can be used to find which shapes are around others. The outer most shape is the boundary.
4. "Corners of the boundaries are identified… by filtering the (x,y) coordinates of the boundaries and looking for peaks in curvature. This yields a set of quadrilaterals (by removing those shapes that do not have four corners)." Stein, 2017
5. A perspective transformation is computed for the square (based on the corners), using homography ("which is a mathematical specification of the perspective transformation"). This tells how tilted the square is.
6. The list of squares is filtered, to keeping those that are big enough to analyze, and not distorted with a high skew or other asymmetries.

## 62.3. DECODING THE SQUARES

The next step is to decode the symbol. Vector has a set of probe locations within the marker square that it probes for black or white reading. These are usually centered in the cells of a grid.

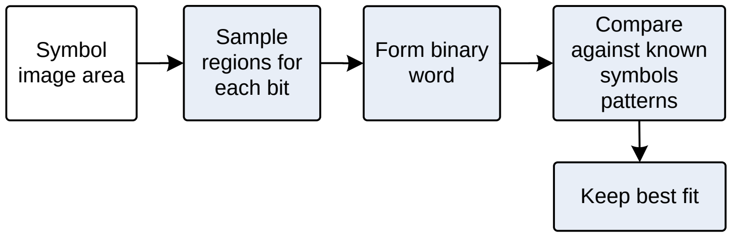



***Figure 46:*** *Decoding the symbol*

The steps in decoding the symbol are:

1. The software uses the perspective transform to map the first point location to one in the image;
2. The pixels at that point in the image are sampled and used to assign a 0 or 1 bit for the sample point.
3. The bit is stored, in a small binary word
4. The above steps are repeated for the rest of the probe locations

This process allows Vector to decode images warped by the camera, its lens, and the relative tilt of the area.

Next, the bit patterns are compared against a table of known symbol patterns. The table includes multiple possible bit patterns for any single symbol, to accommodate the marker being rotated. There is always a good chance of a mistake in decoding a bit. To find the right symbol, Vector:

1. XOR's the decoded bit pattern with each in its symbol table,
2. Counts the number of bits in the result that are set. (A perfect match will have no bits set, a pattern that is off by one bit will have a single bit set in the result, and so on.)
3. Vector keeps the symbol with the *fewest* bits set in the XOR result.

## 62.4. REVAMPING SIZE AND ORIENTATION

The different rotations of the symbol would change the order that it sees the bits. Each bit pattern in the table might also include a note on how much the symbol is rotated (i.e. 0, +90°, -90°, or 180°). When matching a bit pattern, Vector can know the major rotation of the symbol. Combined with the angle of the symbol square, the full rotation of the symbol can be computed.

## 62.5. INFERRING KNOWLEDGE ABOUT OBJECTS

Vector associates an object with symbol. Some objects can have many symbols associated with them. Cubes have different symbols used for sides of cubes. This allows Vector to know what object it is looking at, and what side of the object. And, with some inference, the orientation of the object.

Vector knows (or is told) the physical size of the symbol, and the object holding the symbol. Combining this with the visual size of the object, time of flight distance measurement (if any), and Vector's known position, this allows Vector infer the objects place in the map.

## 63. FACE AND FACIAL FEATURES RECOGNITION

Vector "is capable of recognizing human faces, tracking their position and rotation ("pose") and assigning names to them via an enrollment process." Vector's facial detection and recognition is based on the OKAO vision library. This lets Vector know when one (or more) people are looking at it. This library is primarily used by Vector for facial recognition tasks:

Anki Cozmo SDK

- Face detection ability – the ability to sense that there is a face in the field of view, and locate it within the image.
- Face recognition, the ability to identify whose face it is, looking up the identify for a set of known faces
- Recognize parts of the face, such as eyes, nose and mouth, and where they are located within the image.

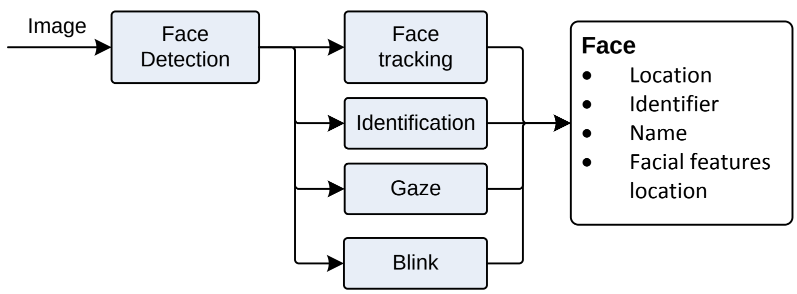



***Figure 47:*** *The face detection and recognition processes*

There are a couple of areas that Vector includes access to in the SDK API, but did not incorporate fully into Vector's AI:

- The ability to recognize the facial expression: happiness, surprise, anger, sadness and neutral. This is likely to be unreliable; that is the consensus of research on facial expression software.
- Ability to estimate the direction of gaze

And there are several features in OKAO that are not used

- The ability to estimate the gender and age of the person
- Human upper body detection
- Hand detection and the ability to detect an open palm. The hand detection used in Vector is done in a different way (which we will discuss in a section below.)

### 63.1. FACE DECTION

OpenCV also has facial detection, but not recognition. OpenCV's classic face detector is an implementation of an algorithm developed by Viola-Jones. Since we know how that works, we can discuss it as representative of how OKAO may work. Viola-Jones applies a series of fast filters (called a "cascade" in the jargon) to detect low-level facial features (called Haar feature selection) and then applies a series of classifiers (also called a cascade). This divides up interesting areas of the image, identify facial parts, and makes conclusions about where a face is.

Vector's face detector (and facial recognition) can't tell that it is looking at an image of face – such as a picture, or on a computer screen – rather than an actual face. One thing that Anki was considering for future products was to move the time of flight sensor next to the camera. This

Daniel Casner, 2019

would allow Vector to estimate the size of the face (and its depth variability) but measuring the distance.

Side note: Anki was exploring ideas (akin to the idea of object permanence) to keep track of a known person or object in the field of view even when it was too small to be recognized (or detected).

## 63.2. FACE IDENTIFICATION AND TRAINING

When it sees a face, it forms a description of the facial features using twelve points:

- Each eye has three points,
- The nose has two,
- The mouth has four points

If you introduce yourself to Vector by voice, you are permitting the robot to associate the name you provide with Facial Features Data for you. Facial Features Data is stored with the name you provide, and the robot uses this data to enhance and personalize your experience and do things like greet you by that name. This data is stored locally on the robot and in the robot's app. It is not uploaded to Anki nor shared, and you can delete it anytime.

## 63.3. COMMUNICATION INTERFACE

There are several commands to manage the faces that Vector recognizes, and to keep informed of the faces that Vector sees. See Chapter 13 for more details.

- The *Enable Face Detection* command enables and disables face detection and analysis stages.
- The *RobotChangedObservedFaceID* and *RobotObservedFace* events are used to indicate when a face is detected, and tracking it: the identity of the face (if known), where it is in the field of view, the facial expression, where key parts of the face are (in the view), etc
- The *Set Face to Enroll* command is used to ability assign a name to face, and the *Update Enrolled Face by Id* command is used to change the name of a known face
- The *Request Enrolled Names* command is used to retrieve a list the known faces
- The ability to remove a facial identity (see *Erase Enrolled Face By Id*), or all facial entities (see *Erase All Enrolled Faces*)
- The *Find Faces* command initiates the search for faces

## 64. TENSORFLOW LITE, DETECTING HANDS, PETS… AND THINGS?

Vector includes support to detect hands, and has preliminary support for detecting pets and a wide variety of objects. These are done using TensorFlow Lite[40], an inference only neural-net discriminator.

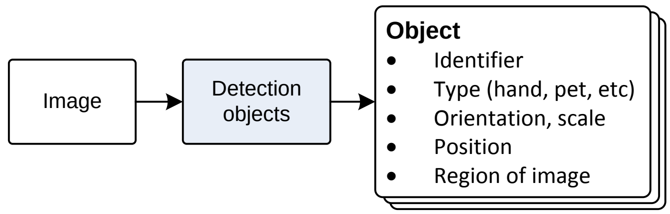



***Figure 48:*** *The processing of the image for symbols and objects*

Vector's hand detection is done with a custom TensorFlow Lite DNN model. Vector also includes the stock MobileNet V1 (0.5, 128) model to classify images, although it does not appear to have been used yet. This model was likely intended to give Vector the ability to identify a wide variety of things, and pets.[41]

***Figure 49:*** *Vector recognizing fruit drawing by Jesse Easley*

MobileNet V1 includes higher quality models that may be explored. Since this model was released, a version 2 and version 3 of MobileNet have been developed and released. These may be faster, higher quality, and/or require fewer processor resources.

The TensorFlow Lite framework is modularly designed. It allows loading different classification trees (models), and being configured to using fast, processor-specific implementations of key kernels.

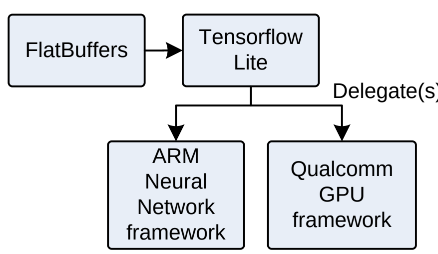



***Figure 50:*** *TensorFlow lite with hardware specific accelerators*

[40] Since both Tensorflow Lite was both introduced at the end of 2017, there has been a steady trickle of improvements to TensorFlow Lite. There is a lower power version that targets microcontrollers.
[41] Or a special model for recognizing pets may have been under development

# 65. PHOTOS/PICTURES

Vector has the ability to take pictures. These pictures are stored on Vector, not in the cloud. The mobile application and SDK applications can view, delete or share pictures taken by Vector.

## 65.1. COMMUNICATION INTERFACE

There are several commands to manage the photographs that Vector has taken. See Chapter 13 for more details.

- The *Photos Info* command is used to retrieve a list of the photographs that Vector currently has
- The *Photo* command is used to retrieve a photo
- The *Delete Photo* command removes a photo from the system
- The *Thumbnail* command retrieves a small version of the image, suitable for displaying as a thumbnail

# 66. RESOURCES & RESOURCES


ARM, Neural-network Machine learning software repo
https://github.com/ARM-software/armnn

Barrett, L. F., Adolphs, R., Marsella, S., Martinez, A. M., & Pollak, S. D. Emotional *expressions reconsidered: Challenges to inferring emotion from human facial movements*. Psychological Science in the Public Interest, 20, 1–68. (2019). doi:10.1177/1529100619832930
https://journals.sagepub.com/stoken/default+domain/10.1177%2F1529100619832930-FREE/pdf

This paper describes the limitations and high error rate of facial expression software.

FloydHub, *Teaching My Robot With TensorFlow*, 2018 Jan 24,
https://blog.floydhub.com/teaching-my-robot-with-tensorflow/

Google, *MobileNets: Open-Source Models for Efficient On-Device Vision, 2017*, Jun 14,
https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

Hollemans, Matthijs. *Google's MobileNets on the iPhone*, 2017 Jun 14
https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

*MobileNet version 2, 2018 Apr 22*
https://machinethink.net/blog/mobilenet-v2/

These two blog posts give an excellent overview of the mechanics of the MobileNet architecture.

Omron, *Human Vision Component (HVC-P2) B5T-007001 Evaluation Software Manual*, 2016
http://www.farnell.com/datasheets/2553338.pdf

Omron, *OKAO Vision Software Library*
https://www.components.omron.com/sensors/image-sensing/solution/software-library

Qualcomm, *How can Snapdragon 845's new AI boost your smartphone's IQ?*, 2018 Feb 1
https://www.qualcomm.com/news/onq/2018/02/01/how-can-snapdragon-845s-new-ai-boost-your-smartphones-iq

Qualcomm, *Snapdragon Neural Processing Engine SDK Reference Guide*,
https://developer.qualcomm.com/docs/snpe/overview.html
Qualcomm Neural Processing software development kit (SDK) for advanced on-device AI, the Qualcomm Computer Vision Suite

Situnayake, Daniel; Pete Warden, *TinyML*, O'Reilly Media, Inc. 2019 Dec,
https://www.oreilly.com/library/view/tinyml/9781492052036/
file:///G:/warehouse/Anki/other/4.%20TensorFlow%20Lite%20for%20Microcontrollers%20-%20TinyML%20[Book].html

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

TensorFlow, Mobile Net v1
https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md
"small, low-latency, low-power models" that can recognize a variety of objects (including animals) in images, while running on a microcontroller

TensorFlow, *TensorFlow Lite GPU delegate*
https://www.tensorflow.org/lite/performance/gpu

TensorFlow, *TensorFlow Lite inference*
https://www.tensorflow.org/lite/guide/inference

This Week in Machine Learning (TWIMLAI), episode 102, *Computer Vision for Cozmo, the Cutest Toy Robot Everrrrr! with Andrew Stein*
https://twimlai.com/twiml-talk-102-computer-vision-cozmo-cutest-toy-robot-everrrrr-andrew-stein/

Viola, Paul; Michael Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features,* Accepted Conference on Computer Vision and Patter Recognition, 2001 http://wearables.cc.gatech.edu/paper_of_week/viola01rapid.pdf

Wikipedia, *Viola-Jones object detection framework* https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework

Viola, Paul; Michael Jones, *Robust Real-time Object Detection,* International Journal of Computer Vision (2001) http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.4868

Example code for running a TensorFlow Lite model on a PC https://github.com/ctuning/ck-tensorflow/blob/master/program/object-detection-tflite/detect.cpp

## CHAPTER 16

# Mapping & Navigation

Vector uses a 2D map to track where he can drive, and where objects are.

- Mapping Overview
- Navigation and Path Planning

## 67. MAPPING OVERVIEW

Vector tracks objects in at least three domains:

- A 2D map that tracks where objects, cliffs, and other things are on the surfaces that he can drive on. This map has an arbitrary origin and orientation
- Vector also tracks where faces and some kinds of recognized objects are in his camera image area; these objects are tracked in the image pixels. (Never mind that the camera pose can change!)
- Vector also has an internal database of symbol-marked objects he knows about, as well faces and pets that he recognizes (or at least is tracking).

Vector's map system builds on the navigation. It uses several sensors to know

- Cliff sensors to detect edge, and lines
- Time of flight sensor to measure distances
- Vision to detect the edges, and the location of a hand
- Vision to identify accessories by recognizing markers

Vector employs a mapping technique known as *simultaneous localization and mapping* (SLAM). Cozmo employ the same (or similar) algorithm:

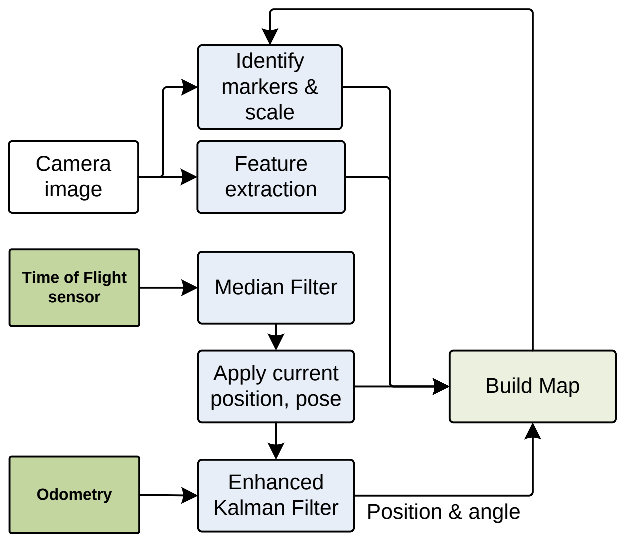



***Figure 51:*** *A typical localization and mapping functional block diagram*

# 68. MAP

Vector keeps tracks of the surface that he can drive on with a 2D map. The maps orientation and position of its origin are arbitrary – Vector just picks a spot and goes with it. The surface map is represented in a compressed format called a quad-tree.

Vector tracks accessory objects, immovable obstacles, and cliffs in terms of this map. The map's units are in mm.

## 68.1. QUAD-TREE MAP REPRESENTATION BASICS

A *quad-tree* is a structured way of "compressing" the information in the map, to reduce the amount of memory required. A quad-tree that recursively breaks down a grid into areas that are interesting, and those that are note.

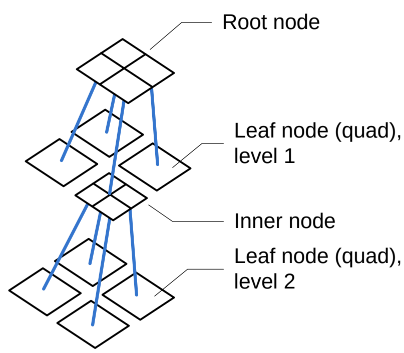



***Figure 52:*** *Structure of a quad-tree*

The tree has two kinds of nodes: inner nodes and leaf nodes.

- The inner nodes do not hold any information about the region (except its size). Instead they point to 4 child nodes at the next lower layer. The top most node is called the *root node.*

- The *leaf nodes* of the tree are square cells (called *quads*) that hold information about what is there (or that the area is unexplored).

Each node represents a square area. The size of the square depends on how many levels it is from the root node. The root node covers the whole map. The nodes in the next layer down are half the width and height of the root node. (In general, a node is half the width and height of a node the next layer up.) *Nodes (including quads) at the same level – the same distance from the root node – are the same size*. Each node's coordinates can be figured in a similar way by knowing the coordinates of the root node.

When Vector reaches the edge of his map area and needs to expand it, he has to add a new node at the top. This becomes the new root node.

## 68.2. WHAT THE MAP TRACKS

Each quad is associated with some information on what is located in that space:

- What Vector knows is in the quad – where there is a cliff, the edge of a line, an object with a marker symbol on it, or an object without a symbol (aka an obstacle).
- Information about that Vector doesn't know about quad – i.e. that he doesn't know whether or not there is a cliff or interesting line edge there
- Whether Vector has visited the quad or not

## 68.3. HOW THE MAP IS SENT FROM VECTOR TO SDK APPLICATIONS

The full tree is not sent from Vector to an SDK, only the leaf nodes (quads). Quads are the only part of the tree that hold information about what is in an area. They also have sufficient information to reconstruct a quad-tree, which is a useful access structure.

## 68.4. TRACKING OBJECTS

Vector tracks objects with markers – and faces, and hands – using the map, and possibly a cross-referencing structure. Vector associates the following information with each object it tracks:

- The object's kind (dock, cube, etc)
- A pose. The image skew gives some partial attitude (relative orientation) information about the symbols and Vector can compute an estimated orientation (relative to the coordinate system) of the object from this and Vector's own pose. Vector can estimate the position his own position, orientation, and the distance measured by the time of flight sensor.
- A size of the object. Vector is told the size of objects with the given symbol.
- A link to a control structure for the kind of object. For instance, accessory cubes can be blinked and sensed

## 69. NAVIGATION AND PLANNING

*Path planning* is devising a path around obstacles without collision, to accomplish some goal, such as docking with the "home" (charger) or accessory cube. Intuitively, all you need to with a rectilinear grid is to figuring out the x-y points to go from point A to B. Vector (and Cozmo) is longer than they are wide – especially when carrying a cube. If this isn't taken into account by the planner, Vector could get stuck going down some path he can't fit in or turn around in. Cozmo had an XY-theta planner to construct paths that he could traverse.

## 70. RESOURCES & RESOURCES

Riisgaard, Søren; Morten Rufus Blas; *SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping*
http://www-inst.eecs.berkeley.edu/~ee290t/fa18/readings/Slam-for-dummies-mit-tutorial.pdf

Wikipedia, *Occupancy grid mapping*,
https://en.wikipedia.org/wiki/Occupancy_grid_mapping

Vector's map is based on occupancy grids, except it does not use probabilities.

CHAPTER 17

# Accessories

This chapter is about Vectors accessories:

- Accessories in general: symbols, docking
- Home & Charging Station
- Companion cube
- Custom items

## 71. ACCESSORIES IN GENERAL

Accessories have at least one maker symbol that Vector can recognize. Vector tracks the location and orientation based on this

### 71.1. DOCKING

Docking is a behaviour/action that is used for both approaching the cube, charging station (home), and other marked items.

It has specialized steps depending on whether it is a cube, the home, etc.

## 72. HOME & CHARGING STATION

Vector has a rich set of behaviours associated with its Home / Charger. In retrospect, this makes sense, as it is Vectors home, his nest, his comfy chair.

### 72.1. DOCKING

Vector's docking with the charging station is:

1. Approach and line up with the charger
2. Turn around (rotate 180°)
3. Reverse and back up the ramp. Vector uses a line follower, with his cliff sensors, to drive straight backwards. (Since he is going backwards, he can't use vision.) He uses the tilt of the ramp to confirm that he is on the charger
4. He also checks that he is in the right spot by looking for power to his charging pads, as reported by base-board charging circuit. If he is unable to find the spot, he grumbles about it, drives off and retries.

Vector has a cute low light mode that turns on most of the pixels on his display to see a bit more, and locate his home.

# 73. COMPANION CUBE

Vector has a companion cube that he can pickup, illuminate the lights on, and detect taps. The cubes design is described in chapter 3.

There is HTTPS API to allow the mobile application and SDK apps to pair Vector with his cube. (Instead of, say, the cube of another Vector.). Vector can be paired with only a single cube.

The HTTPS also allows checking the cubes battery level.

Vector can roll his cube, shove it around, use it to "pop a wheelie," and to pick it up. To do these, he must line up squarely with cube. Vision was found to be needed in the Cozmo to align precisely enough to get the lift hooks into the cube.

# 74. CUSTOM ITEMS

Vector can be told about custom objects. This is three parts. First, the shape of the item – whether it is a "wall" or cube. Second, assign one of a handful of predefined symbols to the item. Cubes can have a symbol for each side. And third, the size of the symbols and object.

Once this is done, Vector can identify the object and track it on his map. These are reported to SDK based applications for their own processing.

## PART V

# Animation

This part describes items that are Vector's animation functions.

- AUDIO PRODUCTION. A look at how Vector's sound effects and how he speaks
- MOTOR CONTROL. A look at how Vector's moves.
- PROCEDURAL FACE. A look at how Vector's moves.
- ANIMATIONS. A look at Vector's scripted motions and sounds.

*drawing by Steph Dere*

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

CHAPTER 18

# Video Display & Face

Vector's LCD is used to display his face, which conveys an emotional connection with his human, his mood, and to display the results of behaviour interactions:

- The image compositor
- The sprite manager
- Animated compositions
- Procedural face

*Note: the animation system is described in a later module*

## 75. OVERVIEW OF THE DIPLAY

Vector's displays imagery – often moving imagery – on his display.

- Full screen sprites — each frame is a PNG image that covers the whole display. A sequence of frames (PNGs) is drawn regularly to create the animated effect.
- Compositor map, with several images scaled and located onto the screen
- Procedural face to draw the face in a complex way (more on this later)

The first two are used as part of behaviors and intents. A visual movie shown when the behavior starts and another is to provide the response. The compositor map allows mixing in iconography, digits and text to show information in the response.

Vector's eyes are drawn in one of two ways

- Using the full-screen sprites above, with the eyes pre-drawn in the PNG's
- Using procedural face which synthesizes the eyes

Note: the sprite and procedural face can be drawn at the same time, with sprites drawn over the eyes. This is done to create weather effects over Vector's face.

### 75.1. ORIGIN

The display system – especially the procedural face module – was pioneered in Cozmo. To prevent burn in and discoloration of the OLED display, Cozmo was given two features. First, Cozmo was given regular eye motion, looking around and blinking. Second, the illuminated rows were regularly alternated to give a retro-technology interlaced row effect, like old CRTs.

US Patent 20372659

Vector's eyes are more refined, but kept the regular eye motion. The interlacing was made optional, and disabled by default.

# 76. RENDERING SYSTEM

The rending system looks like:

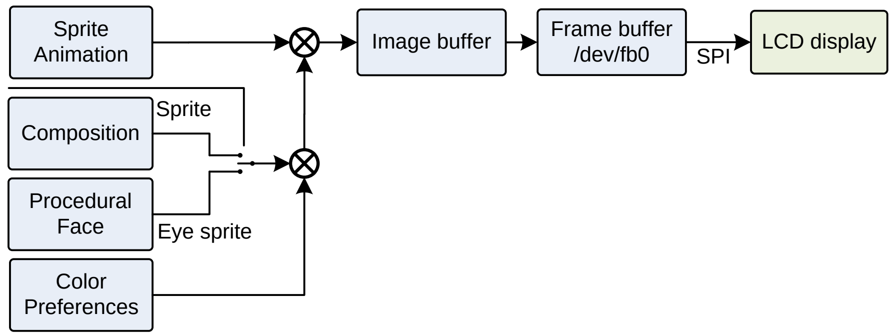



***Figure 53:*** *The display animations functional flow*

## 76.1. SPRITES

Sprites can be displayed on the screen as a sequence. Sprite sequences must either start at 0 and have every frame number, or specify loading via JSON.

If the sprite is grey-scale, it will be colored with the current eye color setting.

## 76.2. COMPOSITION SYSTEM

The screen is composited as a set of boxes (called a sprite box) with a position and size. The composition is given in a JSON file, along with some animated motions.

## 76.3. PROCEDURAL FACE

Vector's dynamic, moving eyes are the gateway for an emotion connection. The procedural face manager is used to animate Vector's eyes. This includes

- The position of each eye
- The center of brightness within each eye (along with the position, this creates the illusion of gaze – that Vector is looking at something)
- The horizontal and vertical scaling of each eye
- A pair of "bends" that represent the cheek, and erase (or occlude) the lower portion of the eyes; these help create the happy emotions
- A pair of "bends" that represent the eye lids and erase (or occlude) the upper portion of the eyes; these help create the sleepy, frustrated/angry emotions.

Eye colour. Vector's eye color is a preference setting, but can be temporarily overridden by the SDK.

CHAPTER 19

# Audio Out

This chapter describes the sound output system:

- The audio output
- Text to speech
- Audio Effects

## 77. SPEAKER

This section describes the audio produced by Vector. Vector uses sound to convey emotion and activities, to speak, and to play sounds streamed from SDK applications and Alexa's remote servers. To support this, it includes a sophisticated audio architecture:

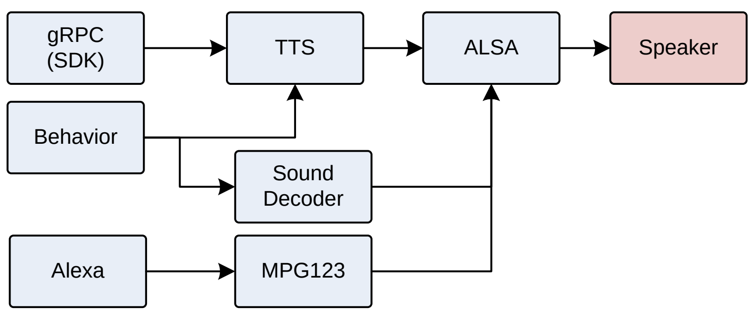



***Figure 54:** The audio output architecture*

Doesn't seem to do sound effect and TTS at the same to time

### 77.1. SOUND FILES

The audio files are located in:

`/anki/data/assets/cozmo_resources/sound`

The audio files are in the WEM/RIFF format, produced by AudioKinetic's WWise tool. The file names are meaningless numbers, with a ".wem" extension.

### 77.2. VOLUME CONTROL

Vector's volume can be set as a setting using the `UpdateSettings` command (see Chapter 13) and the `RobotSettingsConfig` structure (see chapter 22), or using the Master Volume command in Chapter 13. Note: the volume levels using settings doesn't fully match those in the master volume command.

## 77.3. TEXT TO SPEECH

Vector includes a text to speech (TTS) facility. The engine is based on Cozmo text-to-speech subsystem, with the basic software from Acapela. The text to speech engine is part of vic-anim, with some components in libcozmo_engine.

The text-to-speech voices are stored in

`/anki/data/assets/cozmo_resources/tts`

The voice files include:

- co-French-Bruno-22khz
- co-German-Klaus-22khz
- co-Japanese-Sakura-22khz
- co-USEnglish-Bendnn-22khz

### 77.3.1 Configuration

The configuration file for the text to speech engine is located at:

`/anki/data/assets/cozmo_resources/config/engine/tts_config.json`

(This path is hardcoded into vic-anim.) This file is organized as dictionary whose key is the operating system. The "vicos" key is the one relevant for Vector.[42] This dereferences to a dictionary whose key is the language base: "de", "en", "fr", or "ja". The language dereferences to a structure with the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *pitch* | float | Optional. There is a pitch setting field. This is not supported by all voices / platforms. (The comment says that this is Acapela TTS SDK.) "Pitch… adjustment is actually performed by audio layer." |
| *shaping* | | optional |
| *speed* | float | |
| *speedTraits* | array of speedTraits | Optional array speed traits structures (see below) |
| *voice* | string | a path to the ini file within the [assets/cozmo_resources/tts] folder |

***Table 152:** The JSON structure*

Each speedTraits structure has the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *rangeMax* | | |
| *rangeMin* | | |
| *textLengthMax* | | |
| *textLengthMin* | | |

***Table 153:** The speedTraits JSON structure*

[42] The other OS key is "osx" which suggests that Vector's software was development on an OS X platform.

### 77.3.2 Commands

An external application can direct Vector to speak using the Say Text command (see Chapter 13, *Say Text*). The response(s) provide status of where Vector is in the speaking process.

### 77.3.3 Localization

Vector internally has support for German, French, and Japanese, but the application-level language settings only really support US, UK, and Australian dialects of English. The files for non-English localization were not completed.

The localization files for feature stores its text strings (to be spoken) in

/anki/data/assets/cozmo_resources/LocalizedStrings

This path is not present in versions before v1.6. The folder holds sub-folders based on the language:

de-DE en-US fr-FR

Note: there is no ja-JA, but it may be possible to create.

Inside of each are three files intended to provide the strings, for a behaviour, in the locale:

- BehaviorStrings.json
- BlackJackStrings.json
- FaceEnrollmentStrings.json

Each JSON file is a dictionary with the following fields:

***Table 154:*** *The JSON structure*

| Field | Type | Description & Notes |
|---|---|---|
| *smartling* | structure | see below to the structure below. Note all smarting structures examined are the same. |

The dictionary also includes keys, such as "BehaviorDisplayWeather.Rain" that map to a locale specific string. These have the following fields:

***Table 155:*** *The JSON structure*

| Field | Type | Description & Notes |
|---|---|---|
| *translation* | string | The text in the given locale. The string may have placeholders, such as {0}, where text is substituted in. |

Each smartling structure has the following fields:

***Table 156:*** *The smarling JSON structure*

| Field | Type | Description & Notes |
|---|---|---|
| *placeholder_format_custom* | array of strings | An array of patterns that represent possible placeholder patterns. |
| *source_key_paths* | array of strings | "/{*}" Strings are path of a JSON key? |
| *translate_paths* | array of strings | "*/translation" Strings are path of a JSON key? |
| *translate_mode* | string | "custom" |
| *variants_enabled* | boolean | |

This is handled by libcozmo_engine, including the key strings.

### 77.3.3.1 Weather files

The weather behaviour stores its text strings (to be spoken) in

/anki/data/assets/cozmo_resources/config/engine/behaviorComponent/weather/condition_to_tts.json

This path is hardcoded into libcozmo_engine. The JSON file is an array of structures. Each structure has the following fields:[43]

***Table 157:*** *The JSON structure*

| Field | Type | Description & Notes |
|---|---|---|
| *Condition* | string | e.g. "Cloudy", "Cold" |
| *Say* | string | The key used in the BehaviourStrings.json file to look up the localized test. (In previous versions, this was the text to say, in English.) |

### 77.3.4 Customization

Vector's voice files are from Acapela. Acapela sells language packs for book readers, but the format appears different and likely very difficult to modify or create.

Cozmo's employs a different English voice (in the Cozmo APK). This likely could be extracted and used on Vector. (In turn, Vectors voice could probably be used with Cozmo.)

Customization of the Localization TTS would give Vector a bit more personality.

## 78. PARAMETERIC SOUND EFFECTS

Vector has the ability to create sound effects parametrically. The behaviour-animation systems uses this convey Vector's emotional state – sadness, approval, etc. These are like *intersectional tones* that people make – grumbles, and grunting. When an action is being performed, the animation may trigger the sound effect, perhaps to "simulate the physical movement" he is making. The sound effect parameters are modified by the current emotional state, or anticipation – to convey whether he is struggling to do the task, is excited, is frightened, etc.

## 79. REFERENCES AND RESOURCES

Wolford, Jason; Ben Gabaldon, Jordan Rivas, Brian Min *Condition-Based Robots Audio Techniques*, Anki , USPTO Pub.No: US2019/0308327A1, 2018 Apr 6

[43] That this file (and many others) is a simple 1:1 transform lends the suspicion that the localization process is more complex than need be.

# CHAPTER 15

# Motion Control

This chapter describes the motion control subsystem:

- The control of the motors
- Performing head and lift movements
- Moving along paths in a smooth and controlled fashion

## 80. MOTION CONTROL

The motion control is designed to take a path of movements from the path planner or the animation systems. The path consists of arc, line, and turn (in place) movement commands. These can be coordinated with the head and lift, by the animation system.

*Note: the animation system is described in a later chapter*

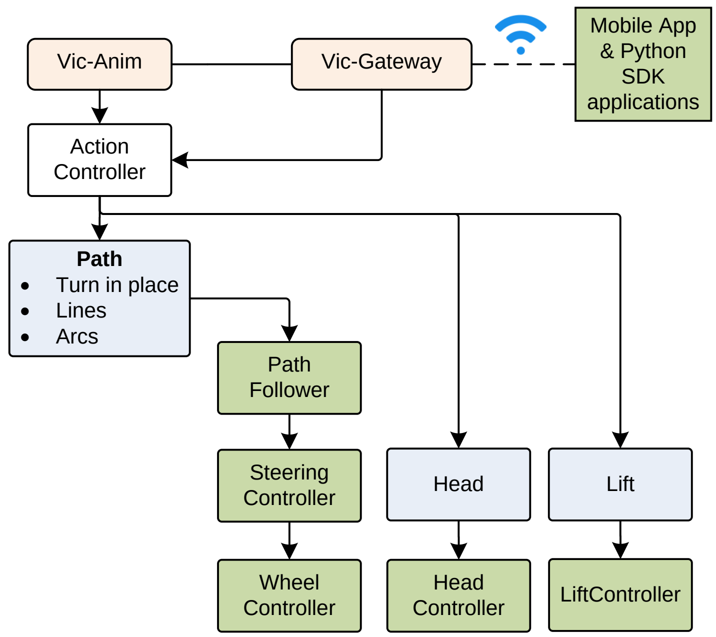



***Figure 55:** Motion controller*

The path planner thinks of the word and robot coordinates within it in terms of x,y and θ (theta) coordinates. The θ being the direction angle that Vector is facing at the time. It builds a list of straight line segments, arcs, and point turns. The PathFollower carries these out. Each of the motors is independently driven and controlled, with the steering controller coordinating the driving actions. Sets gains, executes turns, does docking,

The individual motors have controllers to calibrate, move, prevent motor burnout, and perform any special movements.

## 80.1. FEEDBACK

The motion controller may take position and orientation feedback from

- The linear speed can be estimated from the motors shaft rotation speed (and some estimated tread slip), merged with IMU information
- The speed that the robot is rotating can be measured by the IMU and the vision system.
- The navigation and localization subsystem, which employs a sophisticated Kalman filter on all of the above position.

## 80.2. MOTOR CONTROL

The motor control is (likely) implemented in the base-board, where it can respond most conveniently. A typical speed and position controller for the brushed DC motors is a set of PID control loops. (Although the "d" – derivative – term is often small or not needed.)

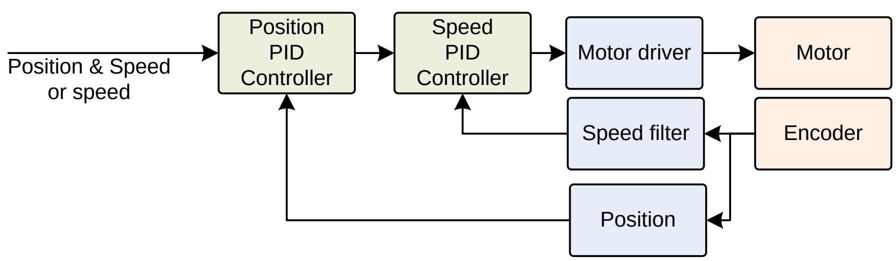



***Figure 56:*** *A typical motor controller*

The motors can be commanded to travel to an encoder position at a speed (given in radians/sec).

The motors can also be "unlocked" – allowed to be spun by external forces. This allows a person to raise and lower the lift, as well as raise and lower the head. Both of these are used as inputs to enter diagnostic modes.

The position – the cumulative number of radians that that the shaft has turned – can be computed by counting encoder events, taking into account the direction the motor was commanded to run. (There is a single encoder for each motor, so it can't determine direction independently.)

The speed of rotation is computed from the encoder counts, taking into account the command direction. The encoder is discrete, and at slow speeds its update rate will produce false measures of shaft speed; the typical way to address this is apply a low-pass filter to the encoder events.

## 80.3. DIFFERENTIAL DRIVE KINEMATICS

Under ideal circumstances, these motions are straight-forward to accomplish:

- To turn in place, the treads turn at the same rate, but in opposite directions. The speed of the turn is proportional to the speed of treads
- To drive straight, both treads turn at the same speed. The speed of motion is proportional to the speed of the treads.
- To drive in an arc is done by driving the treads at two different speeds.

To drive in an arc, the left and right treads are driven speeds:

$$v_{left} = \omega\left(radius + \tfrac{1}{2}\,width\right)$$
$$v_{right} = \omega\left(radius + \tfrac{1}{2}\,width\right)$$

***Equation 3:*** *Tread speeds based on arc radius*

Where

- *width* is Vector's body width
- ω is the angular velocity, i.e. speed to drive around the arc
- radius is the distance from the center of the arc to the center of Vector's body:

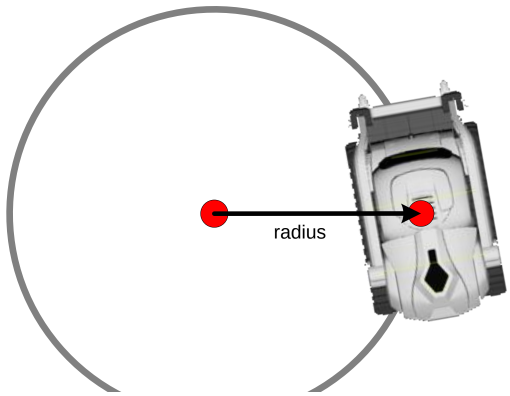



***Figure 57:*** *Radius of arc measurement*

### 80.3.1 Slip

In practice, Vector's actual movement won't quite match what he attempted to do. Mainly this will come from how the treads slip a bit (especially while trying to push an object), and some variation in how driving the motors maps to actual motion.

## 81. MOTION CONTROL COMMANDS

The HTTPS SDK API (Chapter 13) includes commands to control the motors, and to initiate driving actions. The lower level commands, below the action processor are:

- *Drive Wheels*
- *Move Head*
- *Move Lift*
- *Stop All Motors*

The higher level commands, part of the action system are:

- *Drive Straight*
- *Turn In Place*
- *Set Head Angle*
- *Set Lift Height*

# CHAPTER 21

# Animation

This chapter describes Vector's animation engine:

- Animation Engine
- Animation file format

## 82. ANIMATION TRIGGERS AND ANIMATIONS

Vector uses animations, "a sequence of highly coordinated movements, faces, lights, and sounds," "to demonstrate an emotion or reaction."

Vector employs two levels of referring to an animation. Animation, at the lowest level, is the scripted motions (and sounds, etc). Animations are grouped together by type, with an identified called an *animation trigger*. Vector "pick[s] one of a number of actual animations to play based on Vector's mood or emotion, or with random weighting. Thus playing the same trigger twice may not result in the exact same underlying animation playing twice."

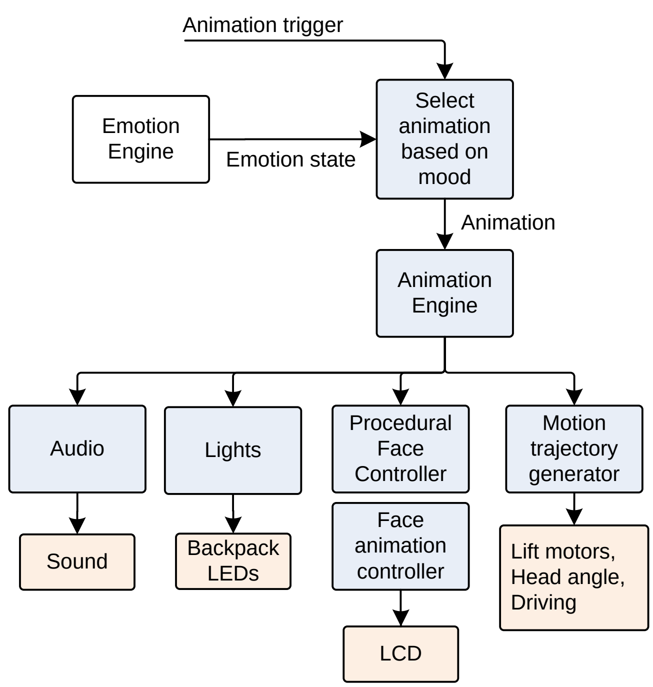



***Figure 58:*** *The behaviour-animation flow*

### 82.1. SDK COMMANDS TO PLAY ANIMATIONS

The HTTPS SDK includes commands to list and play animations.

- A list of animations triggers can be retrieved with the *List Animation Triggers* command.
- A list of animations can be retrieved with the *List Animation* command.

- An animation can be play by selecting the animation trigger (*Play Animation Trigger* command). Vector will select the specific animation from the group. Or,
- An animation can be play by the giving the specific animation name (*Play Animation*).

As the individual animations are low-level they are the most likely to change, be renamed or removed altogether in software updates. Anki strongly recommends using the trigger names instead. "Specific animations may be renamed or removed in future updates of the app."

# 83. ANIMATION BINARY FILE FORMAT

This section describes the schema of the *binary* animation files. (There are other JSON based animation files that are not described in this section.)

## 83.1. OVERVIEW OF THE FILE

The animation file contains one or more named animation "clips." Each clip has one or more tracks that represent the scripted motions (and lights & sounds) that Vector should perform. There are tracks for moving Vector's head, lift, driving, modifying his facial expressions, displaying images on the LCD, audio effects, and controlling the backpack lights.

A clip is composed of key frames (with settings for each of the relevant tracks) that are triggered at different points in time.

## 83.2. RELATIONSHIP WITH COZMO

Vector's file format for animations is derived from the file format used with Cozmo. This presents the possibility of converting Cozmo's animation files to Vector, and vice-versa. They key differences Cozmo and Vector's format are that Vector includes a audio tracks, plus some minor extra fields, and fewer backlight LEDs.

The PyCozmo project has the (experimental) ability extract these animations, and may be useful for this transcoding.

## 83.3. STRUCTURES

The animation file starts with an AnimClips structure. Unless specified otherwise, these are the same as in Cozmo.

### 83.3.1 AnimClips

The AnimClips structure is the "root" type for the file. It provides one or more animation "clips" in the file. Each clip has one or more tracks. The structure following fields:

**Table 158:** *AnimClips structure*

| Field | Type | Description |
|---|---|---|
| *clips* | AnimClip[] | An array of animation clips |

### 83.3.2 AnimClip

The AnimClip has the following fields:

| Field | Type | Description |
|---|---|---|
| *Name* | string | The name of the animation clip |
| *keyFrames* | KeyFrames | The key frames for each of the tracks for this animation clip |

***Table 159:** AnimClip structure*

### 83.3.3 AudioEventGroup

The AudioEventGroup structure is new to Vector. This structure has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *eventIds* | uint[] | | |
| *volumes* | float[] | | |
| *probabilities* | float[] | | |

***Table 160:** AudioEventGroup structure*

### 83.3.4 AudioParameter

The AudioParameter structure is new to Vector. This structure has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *parameterId* | uint | | default:0 |
| *value* | float | | default: 0 |
| *time_ms* | uint | *ms* | default: 0 |
| *curveType* | ubyte | | default: 0 |

***Table 161:** AudioParameter structure*

### 83.3.5 AudioState

The AudioState structure is new to Vector. This structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *switchGroupId* | uint | default: 0 |
| *stateId* | uint | default: 0 |

***Table 162:** AudioState structure*

### 83.3.6 AudioSwitch

The AudioSwitch structure is new to Vector. This structure has the following fields:

| Field | Type | Description |
|---|---|---|
| *switchGroupId* | uint | default: 0 |
| *stateId* | uint | default: 0 |

***Table 163:** AudioSwitch structure*

### 83.3.7 BackpackLights

The BackpackLights structure is used to animate the LEDs on Vector's back. This structure has the following fields:

Table 164: BackpackLights structure

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the backlights are triggered. |
| *durationTime_ms* | uint | *ms* | How long the animation lasts |
| *Front* | float[] | | |
| *Middle* | float[] | | |
| *Back* | float[] | | |

see also: Chapter 8, section *Backpack lights control*

Note: Cozmo's animation structure includes a left and right LED animation.

### 83.3.8 BodyMotion

The BodyMotion structure is used to specify driving motions for Vector. This structure has the following fields:

Table 165: BodyMotion structure

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the motion is triggered. |
| *durationTime_ms* | uint | *ms* | How long the animation lasts |
| *radius_mm* | string | *mm* | TODO: is this the name of a key emotion thing? "TURN_IN_PLACE", "STRAIGHT", possibly a value? |
| *speed* | short | | |

### 83.3.9 Event

The Event structure has the following fields:

Table 166: Event structure

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the motion is triggered. |
| *event_id* | string | | |

### 83.3.10 FaceAnimation

The FaceAnimation structure is used to specify the JSON file to animation Vector's display. This structure has the following fields:

Table 167: FaceAnimation structure

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the motion is triggered. |
| *animName* | string | | |
| *scanlineOpacity* | float | | This is new for Vector. Default: 1.0 |

### 83.3.11 HeadAngle

The HeadAngle structure is used to specify how to move Vector's head. This structure has the following fields:

*Table 168: HeadAngle structure*

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the motion is triggered. |
| *durationTime_ms* | uint | *ms* | How long the animation lasts |
| *angle_deg* | ubyte | *deg* | The angle to move the head to |
| *angleVariability_deg* | ubyte | *deg* | default: 0 |

### 83.3.12 LiftHeight

The LiftHeight structure is used to specify how to move Vector's lift. This structure has the following fields:

*Table 169: LiftHeight structure*

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the motion is triggered. |
| *durationTime_ms* | uint | *ms* | How long the animation lasts |
| *height_mm* | ubyte | *mm* | The height to lift the arms to |
| *heightVariability_mm* | ubyte | *mm* | default: 0 |

### 83.3.13 KeyFrames

The KeyFrames structure the following fields:

*Table 170: KeyFrames structure*

| Field | Type | Description |
|---|---|---|
| *LiftHeightKeyFrame* | LiftHeight[] | |
| *ProceduralFaceKeyFrame* | ProceduralFace[] | |
| *HeadAngleKeyFrame* | HeadAngle[] | |
| *RobotAudioKeyFrame* | RobotAudio[] | |
| *BackpackLightsKeyFrame* | BackpackLights[] | |
| *FaceAnimationKeyFrame* | FaceAnimation[] | |
| *EventKeyFrame* | Event[] | |
| *BodyMotionKeyFrame* | BodyMotion[] | |
| *RecordHeadingKeyFrame* | RecordHeading[] | |
| *TurnToRecordedHeadingKeyFrame* | TurnToRecordedHeading[] | |

### 83.3.14 ProceduralFace

The ProceduralFace structure is used squash, stretch and shake Vectors face in cartoonish ways. It has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the motion is triggered. |
| *faceAngle* | float | | default: 0 |
| *faceCenterX* | float | | default: 0 |
| *faceCenterY* | float | | default: 0 |
| *faceScaleX* | float | | default: 1.0 |
| *faceScaleY* | float | | default: 1.0 |
| *leftEye* | float | | |
| *rightEye* | float | | |
| *scanlineOpacity* | float | | This is new for Vector. default: 1.0 |

***Table 171:*** *ProceduralFace structure*

### 83.3.15 RecordHeading

The RecordHeading structure has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the motion is triggered. |

***Table 172:*** *RecordHeading structure*

### 83.3.16 RoboAudio

The RobotAudio structure is new to Vector; a very different structure with a similar name was used with Cozmo. This structure has the following fields:

| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the audio is triggered. |
| *eventGroups* | AudioEventGroup[] | | |
| *state* | AudioState[] | | |
| *switches* | AudioSwitch[] | | |
| *parameters* | AudioParameter[] | | |

***Table 173:*** *RobotAudio structure*

## 83.3.17 TurnToRecordedHeading

The TurnToRecordedHeading structure has the following fields:

*Table 174: TurnToRecordedHeading structure*

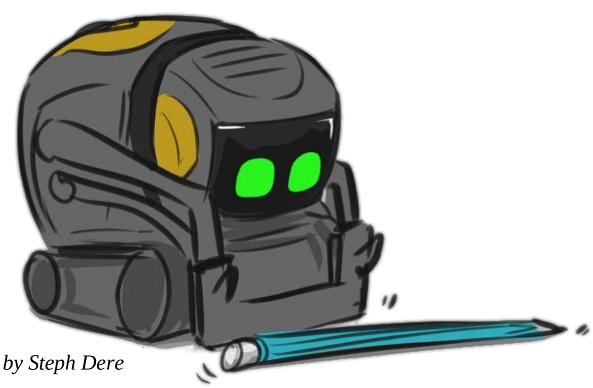
| Field | Type | Units | Description |
|---|---|---|---|
| *triggerTime_ms* | uint | *ms* | The time at which the motion is triggered. |
| *durationTime_ms* | uint | *ms* | How long the animation lasts |
| *offset_deg* | short | *deg* | default: 0 |
| *speed_degPerSec* | short | *deg/sec* | |
| *accel_degPerSec2* | short | *deg/sec$^2$* | default: 1000 |
| *decel_degPerSec2* | short | *deg/sec$^2$* | default: 1000 |
| *tolerance_deg* | ushort | *deg* | default: 2 |
| *numHalfRevs* | ushort | | default: 0 |
| *useShortestDir* | bool | | default: false |

# PART VI

# Maintenance

This part describes practical items to support Vector's operation.

- SETTINGS, PREFERENCES, FEATURES AND STATISTICS. A look at how Vector syncs with remote servers
- SOFTWARE UPDATES. How Vector's software updates are applied.
- DIAGNOSTICS & STATS. The diagnostic support built into Vector, including logging and usage statistics



*drawing by Steph Dere*

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

## CHAPTER 22

# Settings, Preferences, Features, and Statistics

This chapter describes:

- The owner's account settings and entitlements
- The robot's settings (owner preferences)
- The robot's lifetime stats

## 84. THE ARCHITECTURE

The architecture for setting and storing settings, statistics, account information is:

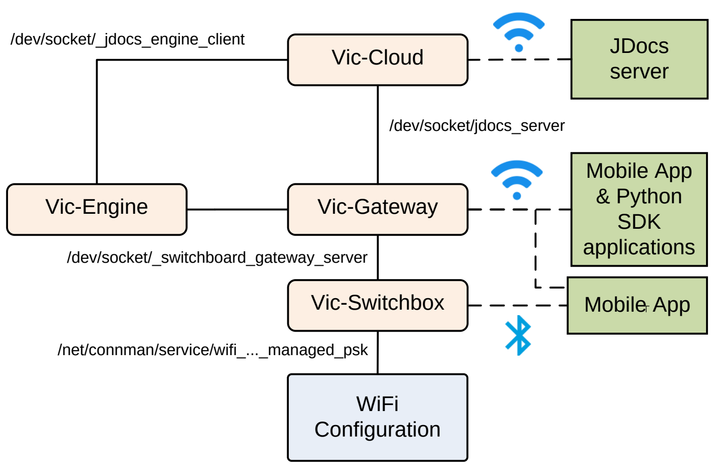



***Figure 59:*** *The architecture for storing preferences, account info, entitlements, and tracking stats*

The Vic-Cloud service accesses information on a remote server.

The Vic-Switchbox interacts with the WiFi subsystem (connman) to allow the mobile App to set the preferred WiFi network to use. The mobile app must use Bluetooth LE to do this.

Vic-Gateway interacts with the mobile App and SDK programs to changes the robot settings.

Vic-Engine receives the preferences from the Vic-Cloud and Vic-Gateway, to carry out an changes in behaviour of Vector.

# 85. WIFI CONFIGURATION

The WiFi configuration (aka settings or preferences) is entirely local to the Vector robot. The information about the WiFi settings is not stored remotely.

The mobile application can configuration the WiFi settings via Vic-Switchbox commands. The WiFi is managed by connman thru the Vic-Switchbox:

- To provide a list of WiFi SSIDs to the mobile app
- To allow the mobile app to select an SSID and provide a password to
- Tell it forget an SSID
- To place the WiFi into Access Point mode

# 86. THE OWNER ACCOUNT INFORMATION

The owner account information is sent from the mobile application to Anki servers at time of registration and setting up a Vector. The owner account information includes:[44]

***Table 175:** The owners account information*

| JSON Key | units | Description & Notes |
|---|---|---|
| *user_id* | base64 | A base64 token to identify the user |
| *created_by_app_name* | string | The name of the mobile application that register the owner. Example: "chewie" |
| *created_by_app_platform* | string | The mobile OS version string when the mobile application created the owners account. Example "ios 12.1.2; iPhone8,1 |
| *created_by_app_version* | string | The version of the mobile application that register the owner. Example: "1.3.1" |
| *deactivation_reason* | | |
| *dob* | YYYY-MM-DD | The owner's date of birth (the one given at time of registration) |
| *drive_guest_id* | GUID | A GUID to identify the owner. This is the same as the "player_id" |
| *email* | string | The email address used to register the account; the same as the user name. |
| *email_failure_code* | | The reason that the email was unable to be verified |
| *email_is_blocked* | boolean | |
| *email_is_verified* | boolean | True if the email verification has successfully completed. False otherwise. |
| *email_lang* | IETF language tag | The IETF language tag of the owner's language preference. example: "en-US" |
| *family_name* | string | The surname of the owner; null if not set |
| *gender* | string | The gender of the owner; null if not set |
| *given_name* | string | The given of the owner; null if not set |
| *is_email_account* | boolean | |
| *no_autodelete* | boolean | |

[44] It is not clear why there is so much information, and why this is sent from the Jdocs server in so many cases.

| | | |
|---|---|---|
| *password_is_complex* | boolean | |
| *player_id* | GUID | A GUID to identify the owner. This is the same as the "drive_guest_id" |
| *purge_reason* | | |
| *status* | string | Example "active" |
| *time_created* | string | The time, in ISO8601 format, that the account was created |
| *user_id* | base64 | A base64 token to identify the owner |
| *username* | string | Same as the email address |

# 87. PREFERENCES & ROBOT SETTINGS

The following settings & preferences are stored in (and retrieved from) the JDoc server. They are set by the mobile app or python SDK program using the HTTPS protocol described in chapter 13. They may also be set (in some cases) by the cloud in response to verbal interaction with the owner, via vic-cloud (e.g. "Hey Vector, set your eye color to teal.").

## 87.1. ENUMERATIONS

### 87.1.1 ButtonWakeWord

When Vector's backpack button is pressed once for attention, he acts as if someone has said his wake word. The ButtonWakeWord enumeration describes which wake word is treated as having been said:

***Table 176:*** *ButtonWakeWord Enumeration*

| Name | Value | Description |
|---|---|---|
| *BUTTON_WAKEWORD_ALEXA* | 1 | When the button is pressed, act as if "Alexa" was said. |
| *BUTTON_WAKEWORD_HEY_VECTOR* | 0 | When the button is pressed, act is "Hey, Vector" was said. |

### 87.1.2 EyeColor

This is the selectable colour to set Vector's eyes to. The JdocType enumeration maps the playful name to the following value used in the RobotSettingsConfig (and vice-versa) and the colour specification:

***Table 177:*** *EyeColor Enumeration*

| Name | Value | Hue | Saturation | Description |
|---|---|---|---|---|
| *CONFUSION_MATRIX_GREEN* | 6 | 0.30 | 1.00 | |
| *FALSE_POSITIVE_PURPLE* | 5 | 0.83 | 0.76 | |
| *NON_LINEAR_LIME* | 3 | 0.21 | 1.00 | |
| *OVERFIT_ORANGE* | 1 | 0.05 | 0.95 | |
| *SINGULARITY_SAPHIRE* | 4 | 0.57 | 1.00 | |
| *TIP_OVER_TEAL* | 0 | 0.42 | 1.00 | |
| *UNCANNY_YELLOW* | 2 | 0.11 | 1.00 | |

The mapping from to enumeration to color values is held in

`/anki/assets/cozmo_resources/config/engine/eye_color_config.json`

(This path is hardcoded into libcozmo_engine.so.) This JSON configuration file is a hash that maps the EyeColor *name* (not the numeric value) to a structure with the "Hue" and "Saturation" values suitable for the SetEyeColor API command. The structure has the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *Hue* | float | The hue to use for the color |
| *Saturation* | float | The saturation to use for the color. |

***Table 178:*** *The eye colour JSON structure*

This structure has the same interpretation as the SetEyeColor request, except the first letter of the keys are capitalized here.

The mapping of the number to the JSON key for the eye colours configuration file is embedded in Vic-Gateway. Adding more named colours would likely require successful complete decompilation and modification. Patching the binary is unlikely to be practical. The colours for the existing names can be modified to give custom, permanent eye colours.

### 87.1.3 Volume

This is the volume to employ when speaking and for sound effects. Note: the MasterVolume API enumeration is slightly different enumeration.

| Name | Value | Description |
|---|---|---|
| *MUTE* | 0 | |
| *LOW* | 1 | |
| *MEDIUM_LOW* | 2 | |
| *MEDIUM* | 3 | |
| *MEDIUM_HIGH* | 4 | |
| *HIGH* | 5 | |

***Table 179:*** *Volume Enumeration*

## 87.2. ROBOTSETTINGSCONFIG

The RobotSettingsConfig structure has the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *button_wakeword* | ButtonWakeWord | When the button is pressed, act as if this wake word ("Hey Vector" vs "Alexa") was spoken.<br>default: 0 ("*Hey Vector*") |
| *clock_24_hour* | boolean | If false, use a clock with AM and PM and hours that run from 1 to 12. If true, use a clock with hours that run from 1 to 24.<br>default: false |
| *default_location* | string | default: "San Francisco, California, United States" |
| *dist_is_metric* | boolean | If true, use metric units for distance measures; if false, use |

***Table 180:*** *The RobotSettingsConfig JSON structure*

| | | |
|---|---|---|
| | | imperial units.<br>default: false |
| *eye_color* | EyeColor | The colour used for the eyes. The colour is referred to by one of an enumerated set. (Within the SDK, the eyes can be set to a colour by hue and saturation, but this is not permanent.)<br>default: 0 (TIP_OVER_TEAL) |
| *locale* | strong | The IETF language tag of the owner's language preference – American English, UK English, Australian English, German, French, Japanese, etc.<br>default: "en-US" |
| *master_volume* | Volume | default: 4 (MEDIUM_HIGH) |
| *temp_is_fahrenheit* | boolean | If true, use Fahrenheit for temperature units; otherwise use Celsius.[45]<br>default: true |
| *time_zone* | string | The "tz database name" for time zone to use for the time and alarms.<br>default: "America/Los_Angeles" |

The default settings are held in

`/anki/assets/cozmo_resources/config/engine/settings_config.json`

(This path is hardcoded into `libcozmo_engine.so`.) The file is a JSON structure that maps each of the fields of `RobotSettingsConfig` to a control structure. The control structure has the following fields:

***Table 181:*** *The setting control structure*

| **Field** | **Type** | **Description & Notes** |
|---|---|---|
| *defaultValue* | | The The value to employ unless one has been given by the operator or other precedent. |
| *updateCloudOnChange* | boolean | true if the value is pushed to the colour when it is changed by the operator. False if not. Won't be restored..? |

It is implied that the setting value is to be pulled from the Cloud when the robot is restored after clearing.

# 88. OWNER ENTITLEMENTS

An entitlement is a family of features or resources that the program or owner is allowed to use. It is represented as set of key-value pairs. This is a concept that Anki provided provision for but was not used in practice.

The only entitlement defined in Vector's API (and internal configuration files) is "kickstarter eyes" (JSON key "`KICKSTARTER_EYES`"). Anki decided not to pursue this, and its feature(s) remain unimplemented.

The default entitlement settings are held in

`/anki/assets/cozmo_resources/config/engine/userEntitlements_config.json`

[45] Anyone else notes that metric requires a true for distance, but a false for temperature? Parity.

(This path is hardcoded into libcozmo_engine.so.) The file is a JSON structure that maps each of the entitlement to a control structure. The control structure is the same as *Table 181: The setting control structure*, used in settings in the previous section.

# 89. VESTIGAL COZMO SETTINGS

There is an "account settings" file held in

`/anki/etc/config/engine/accountSettings_config.json`

This path is hardcoded into libcozmo_engine.so and these settings are only read (possibly) by vic-gateway. The file is a JSON structure that maps each of the settings to a control structure. The control structure is the same as *Table 181: The setting control structure*, used in settings in an earlier section.

The settings include:

***Table 182:** The Cozmo account settings*

| Field | Type | Description & Notes |
|---|---|---|
| *APP_LOCALE* | string | The IETF language tag of the owner's language preference – American English, UK English, Australian English, German, French, Japanese, etc.<br>default: "en-US" |
| *DATA_COLLECTION* | boolean | default: false |

# 90. FEATURE FLAGS

Vector has granular features that can be enabled and disabled thru the use of feature flags. Feature flags allow the code to be deployed, and selectively enabled. As a software engineering practice, a feature is usually is not enabled because the feature is:

- not yet fully developed, or
- specific to a customer, or
- mostly developed and being tested in some groups, or
- only enabled when there is some error occurs or other functionality is not working intended, or
- a special/premium function sold at a cost or reward (like entitlement).

Many of these possibilities do not apply to Anki. But some do. Many of the disabled features are probably disabled because they are incomplete, do not work, and likely not to work for without further development.

## 90.1. CONFIGURATION FILE

The features flag configuration file is located at:

`/anki/data/assets/cozmo_resources/config/features.json`

(This path is hardcoded into libcozmo_engine.so.) This file is organized as an array of structures with the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *enabled* | boolean | True if the feature is enabled, false if not |
| *feature* | string | The name of the feature |

***Table 183:*** *The feature flag structure*

The set of feature flags and their enabled/disabled state can be found in Appendix H. The features are often used as linking mechanisms of the modules. It is likely modules of behavior / functionality.

### 90.2. COMMUNICATION INTERFACE TO THE FEATURES

The list of features can be queried with the `GetFeatureFlagList` command. The status of each individual feature (whether it is enabled or not) can be found with the `GetFeatureFlag` query. See Chapter 13 for more details.

## 91. ROBOT LIFETIME STATISTICS & EVENTS

Vector summarizes his experiences and activities into a set of fun measures. The measures can be found in Appendix J *Table 233: The robot lifetime stats schema*.

The intent is that they can be shared as attaboys and novel dashboard. The lifetime statics 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. The statistics may also be sent the DAS sever.

## 92. REFERENCES & RESOURCES


Wikipedia, *List of tz database time zones*, https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

## CHAPTER 23

# The Software Update process

This chapter describes the software update process

- The software architecture
- The software update process
- How to extract official program files

## 93. THE ARCHITECTURE

The architecture for updating Vector's software is:

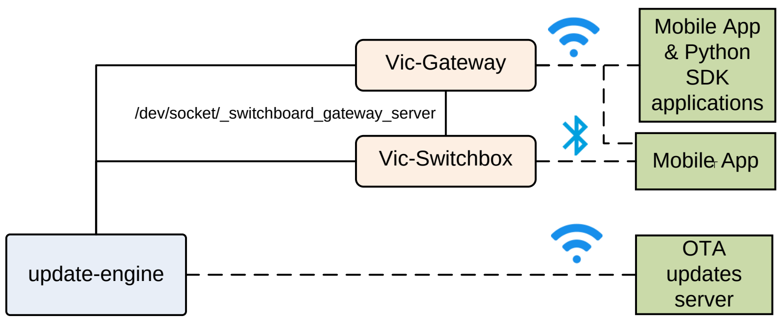



***Figure 60:** The architecture for updating Vector's software*

The Vic-Gateway and Vic-Switchbox both may interact with the mobile App and SDK programs to receive software update commands, and to provide update status information.

The update-engine is responsible for downloading the update, validating it, applying it, and providing status information to Vic-Gateway and Vic-switchbox. [It isn't known yet how they kick off the update]. The update-engine provides status information in a set of files with the "`/run/update-engine`" folder

## 94. THE UPDATE FILE

The update files are TAR files with a suffix "`OTA`". The TAR file has a fixed structure, with some of the files encrypted. There are 3 kinds of update files

- Factory updates. These modify the `RECOVERY` and `RECOVERYFS` partitions.
- Production updates. These modify the `ABOOT`, `BOOT`, and `SYSTEM` partitions
- Delta updates

The archive contains 3 to 5 files, and they must be in a specific order:

1. manifest.ini
2. manifest.sha256
3. apq8009-robot-delta.bin.gz (optionally encrypted). This is present only in delta updates
4. apq8009-robot-emmc_appsboot.img.gz (optionally encrypted). This is present only in factory updates. It will be applied to the ABOOT partition.
5. apq8009-robot-boot.img.gz (optionally encrypted) . This is not present in delta updates. In factory updates it will be applied to the RECOVERY partition; otherwise it will be applied to the BOOT partition.
6. apq8009-robot-sysfs.img.gz (optionally encrypted). In factory updates it will be applied to the RECOVERYFS partition; otherwise it will be applied to the SYSTEM partition.

## 94.1. MANIFEST.INI

The manifest.ini is checked by verifying its signature[46] against manifest.sha256 using a secret key (/anki/etc/ota.pub):

```
openssl dgst \
   -sha256 \
   -verify /anki/etc/ota.pub \
   -signature /run/update-engine/manifest.sha256 \
   /run/update-engine/manifest.ini
```

***Example 7:** Checking the manifest.ini signature*

*Note: the signature check that prevents turning off encryption checks in the manifest below. At this time the signing key is not known.*

All forms of update have a [META] section. This section has the following structure:

| Key | Description |
|---|---|
| *ankidev* | 0 if production release, 1 if development |
| *manifest_version* | Acceptable versions include 0.9.2, 0.9.3, 0.9.4, 0.9.5, or 1.0.0 |
| *num_images* | The number of img.gz files in the archive. The number must match that of the type of update file it is. 1, 2, or 3 |
| *qsn* | The Qualcomm Serial Number; if there are three images (ABOOT, RECOVERY, RECOVERYFS) present, the software is treated as a factory update. The QSN must match the robot's serial number. Optional. |
| *reboot_after_install* | 0 or 1. 1 to reboot after installing. |
| *update_version* | The version that the system is being upgrade to, e.g. 1.6.0.3331 |

***Table 184:** manifest.ini META section*

After the [META] section, there are 1 to 3 sections, depending on the type of update:

- A delta update has a [DELTA] section
- A regular update has a [BOOT], [SYSTEM] sections; both must be present/

[46] I'm using the information originally at: https://github.com/GooeyChickenman/victor/tree/master/firmware

- A factory update has [ABOOT], [RECOVERY], and [RECOVERYFS] sections; all 3 must be present.

Each of these sections has the same structure:

*Table 185: manifest.ini image stream sections*

| Key | Description |
|---|---|
| *base_version* | The version that Vector must be at in order to accept this update. Honored only in delta updates. |
| *bytes* | The number of bytes in the uncompressed archive |
| *compression* | gz (for gzipped). This is the only supported compression type. |
| *delta* | 1 if this is a delta update; 0 otherwise |
| *encryption* | 1 if the archive file is encrypted; 0 if the archive file is not encrypted. |
| *sha256* | The digest of the decompressed file must match this |
| *wbits* | 31. Not used buy update-engine |

## 94.2. HOW TO DECRYPT THE OTA UPDATE ARCHIVE FILES[47]

How to decrypt the OTA update archive files:

*Example 8: Decrypting the OTA update archives*

```
openssl enc -d -aes-256-ctr -pass file:ota.pas -in apq8009-robot-boot.img.gz -out
apq8009-robot-boot.img.dec.gz
openssl enc -d -aes-256-ctr -pass file:ota.pas -in apq8009-robot-sysfs.img.gz -out
apq8009-robot-sysfs.img.dec.gz
```

To use OpenSSL 1.1.0 or later, add "-md md5" to the command:

*Example 9: Decrypting the OTA update archives with Open SSL 1.1.0 and later*

```
openssl enc -d -aes-256-ctr -pass file:ota.pas -md md5 -in apq8009-robot-boot.img.gz -
out apq8009-robot-boot.img.dec.gz
openssl enc -d -aes-256-ctr -pass file:ota.pas -md md5 -in apq8009-robot-sysfs.img.gz -
out apq8009-robot-sysfs.img.dec.gz
```

Note: the password on this file is insecure (ota.pas has only a few bytes[48]) and likely intended only to prevent seeing the assets inside of the update file. The security comes from (a) the individual image files are signed (this is checked by the updater), and (b) the file systems that they contain are also signed, and are checked by aboot and the initial kernel load. See Chapter 6 *Startup* for the gory details.

Signing the files is a whole other kettle of fish.

[47] https://groups.google.com/forum/#!searchin/anki-vector-rooting/ota.pas%7Csort:date/anki-vector-rooting/YlYQsX08OD4/fvkAOZ91CgAJ
[48] Opening up the file in a UTF text editor will show Chinese glyphs; google translate reveals that they say "This is a password"

# 95. THE UPDATE PROCESS

## 95.1. STATUS DIRECTORY

The update-engine provides its status thru a set of files in the `/run/update-engine` folder.

**Table 186:** *update-engine status file*

| File | Description |
|---|---|
| *done* | If this file exists, the update has completed |
| *error* | The error code representing why the update failed. See Appendix D, *Table 204: OTA update-engine status codes*. |
| *expected-download-size* | The expected file size (the given total size of the OTA file) to download. |
| *expected-size* | In non-delta updates, the total number of bytes of the unencrypted image files. This is the sum of the "bytes" field in the sections. |
| *progress* | Indicates how many of the bytes to download have been completed, or how much of the partitions have been written. |

This folder also holds the unencrypted, uncompressed files from the OTA file:

- `manifest.ini`
- `manifest.sha256`
- `delta.bin`
- `aboot.img`
- `boot.img`

## 95.2. PROCESS

The update process if there is an error at any step, skips the rest, deletes the bin and img files.

1. Remove everything in the status folder
2. Being downloading the OTA file. It does *not* download the TAR then unpack it. The file is unpacked as it is received.
3. Copies the manifest.ini to a file in the status folder
4. Copies the manifest.sha256 to a file
5. Verifies the signature of the manifest file
6. Validates that the update to the OTA version is allowed. This includes a check that it is to a newer version number, and the developer vs production software type matches whether this Vector is a developer or production.
7. If this is factory update, it checks that the QSN in the manifest matches Vector's QSN.
8. It marks the target partition slots as unbootable
9. Checks the img and bin contents
    a) delta
    b) boot & system

c) If this is a factory update, aboot, recovery, and recoveryfs

10. If this is a factory update:

    a) Sets /run/wipe-data. This will trigger erasing all of the user data on the next startup

    b) Makes both a and b slots for BOOT and SYSTEM partitions as unbootable

11. If this is not factory update

    a) Sets the new target slot as active

12. Deletes any error file

13. Sets the done file

14. Posts a DAS event robot.ota_download_end to success + next version

15. If the reboot_after_install option was set, reboots the system

# 96. RESOURCES & RESOURCES

https://source.android.com/devices/bootloader/flashing-updating
Describes the a/b process as it applies to android

# CHAPTER 24

# Diagnostics

This chapter describes the diagnostic support built into Vector

- The customer care information screen
- The logging of regular use
- Crash logs
- Gathering usage, and performance data

## 97. OVERVIEW

Anki gathers "analytics data to enable and improve the services and enhance your gameplay… Analytics Data enables us to analyze crashes, fix bugs, and personalize or develop new features and services." There are lot different services that accomplish the analytics services. This data is roughly: logs, crash dumps and "DAS manager"

Logging and diagnostic messages are typically not presented to the owner, neither in use with Vector or thru the mobile application... nor even in the SDK.

The exception is gross failures that display a 3-digit error code. This is intended to be very exceptional.

Diagnostic and logging information is available thru undocumented interfaces.[49]

### 97.1. THE SOFTWARE INVOLVED

There are many different programs and libraries used in the diagnostic and logging area. The table below summarizes of them:

**Table 187:** *Vector diagnostic & logging software*

| Program / Library | Description |
|---|---|
| *animfail* | This is started by the animfail service. |
| *anki-crash-log* | Copies the last 500 system messages and the crash dump passed to the command line to a given log file. This is called by vic-cloud, vic-dasmgr, vic-engine, vic-gateway, vic-log-kernel-panic, vic-log-upload, vic-robot, vic-switchboard, and the anki-crash-log service. |
| *ankitrace* | This program wraps the Linux tract toolkit (LTTng). This program is not present in Vector's file system. This is called by fault-code-handler. |
| *cti_vision_debayer* | This is not called. |
| *diagnostics-logger* | Bundles together several log and configuration states into a compressed tar file. This is called by vic-switchboard, in a response to a Bluetooth LE log command. |

[49] The lack of documentation indicates that this was not intended to be supported and employed by the public... at least not until other areas had been resolved.

| | |
|---|---|
| *displayFaultCode* | Displays error fault codes on the LCD. This is not called; see vic-faultCodeDisplay. |
| *fault-code-clear* | This clears and pending or displayed faults (by deleting the relevant files). This allows new fault code to be displayed. This is called by vic-init.sh. |
| *fault-code-handler* | XXX This is called by the fault-code service. |
| *librobotLogUploader.so* | Sends logs to cloud. This library is employed by libcozmo_engine, vic-gateway and vic-log-upload. |
| *libosState* | Used to profile the CPU temperature, frequency, load; the WiFi statistics, and ETC. This is used by libvictor_web_library, vic-anim, and vic-dasmgr. |
| *libwhiskeyToF* | This unusually named library[50] has lots of time of flight sensor diagnostics. This is present only in version 1.6. This library is employed by libcozmo_engine. |
| *rampost* | This is to perform initial communication and version check of the firmware on the baseboard (syscon). This exists within the initial RAM disk, and is called by init. |
| *vic-anim* | Includes the support for the Customer Care Information Screen. This is started by the vic-anim service. |
| *vic-crashuploader-init* | Removes empty crash files, renames the files ending in ".dmp~" to ".dmp". This is called by the vic-crashuploader service. |
| *vic-crashuploader* | A script that sends crash mini-dump files to backtrace.io. This is called by the vic-crashuploader service. |
| *vic-dasmgr* | This is started by the vic-dasmgr service. |
| *vic-faultCodeDisplay* | Displays error fault codes on the LCD. This is called by fault-code-handler. |
| *vic-init.sh* | Takes the log messages from rampost and places then into the system log, forwards any kernel panics. This is started by the vic-init service. |
| *vic-log-event* | A program that is passed an event code in the command line. This is called by TBD. |
| *vic-log-forward* | This is called by vic-init.sh |
| *vic-log-kernel-panic* | This is called by vic-init.sh |
| *vic-log-upload* | This is called by vic-log-uploader |
| *vic-log-uploader* | "This script runs as a background to periodically check for outgoing files and attempt to upload them by calling 'vic-log-upload'." This is started by the vic-log-uploader service. |
| *vic-logmgr-upload* | "This script collects a snapshot of recent log data" into a compressed (gzip) file, then uploads the file" and software revision "to an Anki Blobstore bucket." This is not called. |
| *vic-on-exit* | Called by systemd after any service stops. This script places the fault code associated with the service (if another fault code is not pending) into /run/fault_code for display. |
| *vic-powerstatus.sh* | Record every 10 seconds the CPU frequency, temperature and the CPU & memory usage of the "vic-" processes. This is not called. |

(Quotes from Anki scripts.) Support programs are located in /bin, /anki/bin, and /usr/bin

[50] Anki has taken great care for squeaky-clean image, even throughout the internal files, so it's a surprise to see one clearly named after a rude acronym (WTF).

# 98. SPECIAL SCREENS AND MODES

Vector has 3 special screens and two special modes. The screens are

- Customer Care Info Screen (CCIS) that can display sensor values and other internal measures,
- Debug screen used to display Vector's serial number (ESB) and IP address, and
- The fault code display which is used to display a 3-digit fault code when there is an internal failure (this screen is only displayed if there is a fault, and can't be initiated by an operator.)

Vector has two special modes

- Entering recovery mode, to force Vector use factory software and download replacement firmware. (This mode doesn't delete any user data.)
- "Factory reset" which erases all user data, and Vector's robot name

## 98.1. CUSTOMER CARE INFORMATION SCREEN

Customer Care Info Screen (CCIS). It has a series of screens that display sensor values and other readings.

See https://www.kinvert.com/anki-vector-customer-care-info-screen-ccis/ for a walk thru

## 98.2. VECTORS' DEBUG SCREEN (TO GET INFO FOR USE WITH THE SDK)

Steps to enter the debug screen

1. Place Vector on the charger,
2. Double-click his backpack button,
3. Move the arms up and down

This will display his ESN (serial number) and IP address. The font is much smaller than normal, and may be hard to read.

## 98.3. DISPLAY FAULT CODES FOR ABNORMAL SYSTEM SERVICE EXIT / HANG

If there is a problem while the system is starting or running– such as one of the services exits (e.g. crashes) – a fault code associated with that service is stored in `/run/fault_code` and the fault code displayed. See Appendix D for fault codes.

See chapter XXXX

## 98.4. RECOVERY MODE

Vector includes a *recovery mode* that is used to force Vector to boot using factory software. The recovery mode will not delete any user data or software that had previously been installed via Over-The-Air (OTA) update.

The recovery mode is intended to help with certain issues such as Vector failing to boot up using the regular firmware. He may have been unable to charge (indicated by teal Back Lights), or encountered other software bugs[51].

The application in the recovery mode attempts to download and reinstall the latest software. This is likely done under the assumption that the firmware may be corrupted, or not the latest, and that a check for corruption would take so long as to not be useful.

### 98.5. "FACTORY RESET"

Erases all user data, include pictures, faces, and API certificates. It clears out the robot name. The Vector will be given a new robot name when he is set up again.

The name "factory reset" is controversial, as this does not truly place Vector into an identical software state as robot in the factory.

## 99. BACKPACK LIGHTS

The lights on the backpack are primarily set by Vic-robot, but driven by the base-board. If the base-board firmware (syscon) is unable to communicate with Vic-robot, it will set the lights on its own.

## 100. DIAGNOSTIC COMMANDS

There are several HTTPS commands that are useful for diagnosing errors:

The connectivity with the cloud can be checked to see if the servers can be reached, if the authentication (i.e. username and password) is valid, if the server certificate is valid. See Chapter 13, *Check Cloud Connection*.

The debug logs can be requested to be sent to the server for analysis. See the Upload Debug Logs command in Chapter 13, *UploadDebugLogs*

## 101. LOGS

- Logs can be downloaded to a PC or mobile application using the Bluetooth LE API
- The Logs can be used to the server using the SDK command X

### 101.1. GATHERING LOGS, ON DEMAND

The logs can be requested by issuing a log fetch command via Bluetooth LE. Vic-switchboard handles the request, delegating the preparation of the log files to diagnostics-logger. This utility gathers the following tars and compresses them:

***Table 188:*** *Files in the log archive*

| File | Description |
|---|---|
| *connman-services.txt* | connmanctl services |
| *dmesg.txt* | dmesg |
| *ifconfig.txt* | ifconfig wlan0 |
| *iwconfig.txt* | iwconfig wlan0 |

[51] The web page says that are "indicated by a blank screen. If you get a status code between 200-219, recovery mode will also help."

| | |
|---|---|
| *log.txt* | Concenates /var/log/messages.1.gz (uncompressed) and /var/log/messages |
| *netstat-ptlnu.txt* | netstat -ptlnu |
| *ping-anki.txt* | Ping's anki.com for connectivity and latency. |
| *ping-gateway.txt* | Looks up the IP address (using netstat) of the gateway that Vector is using and pings it for connectivity and latency. |
| *ps.txt* | Process stats (ps) of Anki's "Vic" processes |
| *top.txt* | top -n 1 |

This utility is triggered by:

- vic-switchboard when issued a log fetch command (via Bluetooth LE).
- Vic-gateway when the upload log command is issued
- Other

# 102. CRASHES

Crash logs are sent on system start (reboot). They are primarily minidump files produced by Google breakpad, and are sent to backtrace.io for analysis.

# 103. CONSOLE FILTER

The logging by functional blocks (primarily in Vic-engine) can be configured. The logging configuration file is located at:

`/anki/data/assets/cozmo_resources/config/engine/console_filter_config.json`

This file is organized as dictionary whose key is the operating system. The "vicos" key is the one relevant for Vector.[52] It dereferences to a structure with the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *channels* | array | An array of the channel logging enable structures |
| *levels* | array | An array of logging level enable structures |

***Table 189:** The console filter channel structure*

This "channels" is as an array of structures with the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *channel* | string | The name of the channel |
| *enabled* | boolean | True if should log information from the channel, false if not. |

***Table 190:** The channel logging enable structure*

This "levels" is an array of structures with the following fields:

| Field | Type | Description & Notes |
|---|---|---|
| *enabled* | boolean | True if should log information at that level, false if not. |

***Table 191:** The logging level enable structure*

[52] The other OS key is "osx" which suggests that Vector's software was development on an OS X platform.

| | | |
|---|---|---|
| *level* | string | "event" or "debug" |

The features are used as linking mechanisms of the modules. It is likely modules of behavior / functionality. It is not clear how it all ties together.

***Table 192:** The channels*

| Channel | enabled | Description & Notes |
|---|---|---|
| *Actions* | false | |
| *AIWhiteboard* | false | |
| *Alexa* | false | |
| *Audio* | false | |
| *Behaviors* | false | |
| *BlockPool* | false | |
| *BlockWorld* | false | |
| *CpuProfiler* | true | |
| *FaceRecognizer* | false | |
| *FaceWorld* | false | |
| *JdocsManager* | true | |
| *MessageProfiler* | true | |
| *Microphones* | false | |
| *NeuralNets* | false | |
| *PerfMetric* | true | |
| *SpeechRecognizer* | false | |
| *VisionComponent* | false | |
| *VisionSystem* | false | |
| * | false | |

# 104. USAGE STUDIES AND PROFILING DATA

Anki had ambitious to perform engagement studies and experiments with device settings:

> "The Services collect gameplay data such as scores, achievements, and feature usage. The Services also automatically keep track of information such as events or failures within them. In addition, we may collect your device make and model, an Anki-generated randomized device ID for the mobile device on which you run our apps, robot/vehicle ID of your Anki device, ZIP-code level data about your location (obtained from your IP address), operating system version, and other device-related information like battery level (collectively, "Analytics Data")."

The DAS manager protocol's version identifier dates to development of Overdrive. One patent on their "Adaptive Data Analytics Service" is quite an ambitious plan to tune an improve systems.

> "A closed-loop service, referred to as an Adaptive Data Analytics Service (ADAS), characterizes the performance of a system or systems by providing information describing how users or agents are operating the system, how the system components interact, and how these respond to external influences and factors. The ADAS then builds models and/or defines relationships that can be used to optimize performance and/or to predict the results of changes made to the system(s). Subsequently, this learning provides the basis for administering, maintaining, and/or adjusting the system(s) under study. Measurement can be ongoing, even after the operating parameters or controls of a system under the administration or monitoring of the ADAS have been adjusted, so that the impact of such adjustments can be determined. This recursive process of observation, analysis, and adjustment provides a closed-loop system that affords adaptability to changing operating conditions and facilitates self-regulation and self-adjustment of systems."

There is no information on whether this was actually accomplished, or that these techniques were used in Cozmo or Vector. Anki developed "both batch and real-time dashboards to gain insights over device and user behavior," according to their Elemental toolkit literature.

## 104.1. EVENT TRACING

The DAS manager on Vector and the mobile application posts event such as when an activity begins, key milestones along the way, and when the activity ends. The events can include extra parameters such as text and values. In the case of the mobile application, this is the name of each button pressed, screen displayed, error encountered, and so forth.

Speculated purpose:

- To identify how far people got in a process, or what their flow thru an interaction is
- To estimate durations of activities, such as onboarding, how long Vector can play between charge cycles, and how long a charge cycle is.
- To identify unusual events (such as errors).
- May allow detailed reconstruction of the setup, configuration and interaction

The event naming pattern is [*module name*].[*some arbitrary name*]

When these are logged in Vector's text log files they are prefixed with an '@' symbol.[53]

For examples of DAS events, see Appendix J.

## 104.2. PROFIILING AND LIBOSSTATE

The tools in Vector gather a variety of diagnostic information about

- Basic information about the robot – the version of software it is running, and what the robot's identifier/serial number is.
- Whether Vector is booted into recovery mode when it is sending the information.
- The uptime – how long Vector has been running since the last reboot or power on.
- The WiFi performance, to understand the connectivity at home since Vector depends so heavily on cloud connectivity for his voice interactions.

[53] This is a very helpful feture

- The CPU temperature profile, to find the balance between overheating and AI performance. Some versions and features of Vector can cause faults due to the processor overheating. Anki probably wanted to identify unusual temperatures and whether their revised settings addressed it.
- The CPU and memory usage statistics for the "vic-" application services. Anki probably sought to identify typical and on unusual processing loads and heavy use cases.
- The condition of the storage system – information about the flash size & partitions, whether the user space is "secure", and whether the EMR is valid.

### 104.2.1 WiFi Stats

libosState gathers the following information about the WiFi network:

- The WiFi MAC address
- The WiFi SSID (and flagged if it isn't valid)
- The assigned IP Address (and flagged if it isn't valid)
- The number of bytes received and sent
- The number of transmission and receive errors

The key files employed to access this information:

***Table 193:*** *The WiFi related stats /proc files*

| File | Description |
|---|---|
| */sys/class/net/wlan0/address* | The IP address assigned to Vector |
| */sys/class/net/wlan0/statistics/rx_bytes* | The number of bytes received |
| */sys/class/net/wlan0/statistics/rx_errors* | The number of receive errors |
| */sys/class/net/wlan0/statistics/tx_bytes* | The number of transmit errors |
| */sys/class/net/wlan0/statistics/tx_errors* | The number of bytes sent |

How this is used: to get a sense of WiFi connectivity in the home, and rooms where Vector is used. Anki's internal research showed that rooms in a home can have a wide range of connectivity characteristics.

Jane Fraser, 2019

### 104.2.2 CPU stats

libosState gathers the following information about the CPU temperature:

- The CPU temperature
- The CPU target and actual frequency
- Whether the CPU is being throttled
- The limits set on the CPU frequency

The key files employed to access this information:

***Table 194:*** *Named device and control files*

| File | Description |
|---|---|
| */sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq* | |
| */sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq* | |
| */sys/devices/system/cpu/cpu0/cpufreq/scaling_governor* | |
| */sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed* | |

| /sys/devices/virtual/thermal/thermal_zone3/temp |
|---|

How this is used: This information was probably intended to find the balance between overheating and AI performance.

## 104.3. EXPERIMENTS

There is an experiments file. This is in libcozmo. Cozmo's APK has a file with the same structure. The file has the following high-level structure:

*Table 195: The experiments structure*

| Field | Type | Description & Notes |
|---|---|---|
| *meta* | meta structure | A structure that describes what project the experiment applies to and the versioning info of the structure. |
| *experiments* | array of experiment structures | An array of experiments, each with their own conditions and parameters. |

The meta structure has the following fields:

*Table 196: The meta JSON structure*

| Field | Type | Description & Notes |
|---|---|---|
| *project_id* | string | "cozmo"[54] |
| *revision* | int | 1 |
| *version* | int | 2 |

An experiment structure has the following fields:

*Table 197: The experiment JSON structure*

| Field | Type | Description & Notes |
|---|---|---|
| *activation_mode* | string | "automatic" |
| *audience_tags* | array of TBD | |
| *forced_variations* | array of TBD | |
| *key* | string | "report_test_auto" |
| *pop_frac_pct* | int | Portion of the population, as a percentage, that will take part in this experiment. |
| *pause_time_utc_iso8601* | string | |
| *resume_time_utc_iso8601* | string | |
| *start_time_utc_iso8601* | string | The date and time that the experiment will commence. |
| *stopt_time_utc_iso8601* | string | The date and time that the experiment will end. |
| *variations* | array of variation | |
| *version* | int | 0 |

A variation structure has the following fields:

[54] I suspect that this would have changed once experiments were initiated with Vector

| Field | Type | Description & Notes |
|---|---|---|
| *key* | string | One of at least two populations subject to the test: "control" or "treatment" |
| *pop_fract_pct* | int | Portion of the population, as a percentage, that will be in this subject group. |

***Table 198:** The variation JSON structure*

# 105. REFERENCES & RESOURCES


Anki, Privacy policy, 2018 Oct 5
https://anki.com/en-us/company/privacy.html

DeNeale, Patrick; Tom Eliaz; *Adaptive data analytics service*, Anki, USPTO US9996369B2, 2015-Jan-05

*os-release — Operating system identification*
https://www.freedesktop.org/software/systemd/man/os-release.html
Describes the `/etc/os-version` and `/etc/os-version-rev` files

# References & Resources

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

## 106. CREDITS

Credit and thanks to Anki, C0RE, Melanie T for access to the flash partitions, file-systems, decode keys, board shots, and information on the motor assembly. Fictiv for board shots. The board shots that help identify parts on the board and inter-connection on the board. HSReina for Bluetooth LE protocol information. Some drawings adapted from Steph Dere, and Jesse Easley's twitter.

## 107. REFERENCE DOCUMENTATION AND RESOURCES

### 107.1. ANKI


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

Casner, Daniel, *Sensor Fusion in Consumer Robots*, Embedded Vision Summit, 2019 May https://www.embedded-vision.com/platinum-members/embedded-vision-alliance/embedded-vision-training/videos/pages/may-2019-embedded-vision-summit-casner https://www.youtube.com/watch?v=NTU1egF3Z3g

Fraser, Jane, *IoT: How it Changes the Way We Test*, Spring 2019 Software Test Professionals Conference, 2019 Apr 3 https://spring2019.stpcon.com/wp-content/uploads/2019/03/Fraser-IoT-How-it-changes-the-way-we-test-updated.pdf

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

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

Stein, Andrew; *Making Cozmo See,* Embedded Vision Alliance, 2017 May 25 https://www.slideshare.net/embeddedvision/making-cozmo-see-a-presentation-from-anki https://youtu.be/Ypz7sNgSzyI


### 107.2. OTHER


cozmopedia.org

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

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

Sriram, Swetha, *Anki Vector Robot Teardown*, Fictiv, 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/

FPL, *FlatBuffers*
https://google.github.io/flatbuffers/

PyCozmo
https://github.com/zayfod/pycozmo/tree/master/pycozmo

Zaks, Mazim *FlatBuffers Explained,* 2016-Jan-30
https://github.com/mzaks/FlatBuffersSwift/wiki/FlatBuffers-Explained

## 107.3. QUALCOMM

Although detailed documentation isn't available for the Qualcomm APQ8009, there is documentation available for the sibling APQ8016 processor.

Qualcomm, *APQ8016E Application Processor Tools & Resources*,
https://developer.qualcomm.com/hardware/apq-8016e/tools

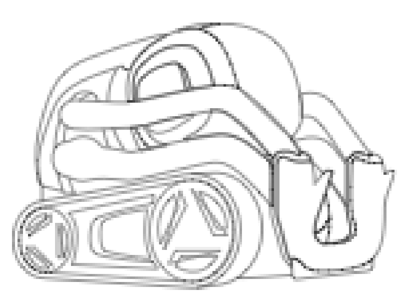
Qualcomm, *DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor ADB Debugging Commands Guide*, LM80-P0436-11, Rev C, 2016 Sep
lm80-p0436-11_adb_commands.pdf

Qualcomm, *DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor Software Build and Installation Guide, Linux Android*, LM80-P0436-2, Rev J, 2016 Dec
lm80-p0436-2_sw-build-and-install_gd_linux_android_dec2016.pdf

# Appendices

These appendices provide extra material supplemental to the main narrative. These include tables of information, numbers and keys.

- 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.
- ALEXA MODULES. This appendix describes the modules used by the Alexa client
- FAULT AND STATUS CODES. This appendix provides describes the system fault codes, and update status codes.
- FILE SYSTEM. This appendix lists the key files that are baked into the system.
- 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
- FEATURES. This appendix enumerates the Vector OS “features” that can be enabled and disabled.
- PHRASES. This appendix reproduces the phrases that the Vector keys off of.
- DAS EVENTS. This appendix describes the identified DAS events
- PLEO. This appendix gives a brief overview of the Pleo animatronic dinosaur, an antecedent with many similarities.



*[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 |
| APQ | application processor Qualcomm (used when there is no modem in the processor module) |
| AVS | Alexa Voice Service |
| BIN | binary file |
| CCIS | customer care information screen |
| CLAD | C-like abstract data structures |
| CNN | convolution neural network |
| CRC | cyclic redundancy check |
| CSI | Camera serial interface |
| DAS | *unknown (diagnostic/data analytics service?)* |
| DFU | device firmware upgrade |
| DTTB | Dance to the beat |
| EEPROM | electrical-erasable programmable read-only memory |
| EMR | electronic medical record |
| ESD | electro-static discharge |
| ESN | electronic serial number |
| FBS | flat buffers |
| FDE | full disc encryption |
| GPIO | general purpose IO |
| GUID | globally unique identifier (effectively same as UUID) |
| HLAI | high-level AI |
| I2C | inter-IC communication |

***Table 199:*** *Common acronyms and abbreviations*

| | |
|---|---|
| **IMU** | inertial measurement unit |
| **IR** | infrared |
| **JDocs** | JSON Documents |
| **JSON** | JavaScript Object Notation |
| **JTAG** | Joint Test Action Group |
| **LCD** | liquid crystal display |
| **LED** | light emitting diode |
| **LUKS** | linux unified key setup |
| **MCU** | microcontroller |
| **mDNS** | multicast domain name service (DNS) |
| **MEMS** | micro-electromechanical systems |
| **MIPI** | mobile industry processor interface |
| **MISO** | master-in, slave-out |
| **MOSI** | master-out, slave-in |
| **MPU** | microprocessor |
| **MSM** | mobile station modem, the APC processor and a modem. |
| **MSRP** | manufacturer's suggest retail price |
| **OLED** | organic light-emitting diode display |
| **OTA** | over the air updates |
| **PCB** | printed circuit board |
| **PCBA** | printed circuit board assembly (PCB with the components attached) |
| **PMIC** | power management IC |
| **PWM** | pulse width modulation |
| **QSN** | Qualcomm serial number |
| **RPM** | resource power management |
| **RRT** | rapidly-expanding random tree |
| **SCLK** | (I2C) serial clock |
| **SDA** | (I2C) serial data |
| **SDK** | software development kit |
| **SLAM** | simultaneous localization and mapping |
| **SOC** | system on a chip |
| **SPI** | serial-peripheral interface |
| **SSH** | secure shell |
| **SSID** | service set identifier (the name of the Wifi network) |

| | |
|---|---|
| **STM32** | A microcontroller family from ST Microelectronics |
| **SWD** | single wire debug |
| **TAR** | tape archive file |
| **TTS** | text to speech |
| **UART** | universal asynchronous receiver/transmitter |
| **USB** | universal serial bus |
| **UUID** | universally unique identifier (effectively same as GUID) |
| **vic** | short for Victor (Vector's working name) |

***Table 200:*** *Glossary of common terms and phrases*

| Phrase | Description | |
|---|---|---|
| *A** | A path finding algorithm | |
| *aboot* | The Android boot-loader used to launch Vector's linux system. | |
| *accelerometer* | A sensor used to measure the angle of Vector's head, and acceleration (change in velocity). | |
| *animation* | "a sequence of highly coordinated movements, faces, lights, and sounds … to demonstrate an emotion or reaction." | |
| *attitude* | Vector's orientation, esp relative to the direction of travel | |
| *beam forming* | A technique using multiple microphones to listen to a single speaker by selectively paying attention to sound only coming from that direction. | |
| *behavior* | "Behaviors represent a complex task which requires Vector's internal logic to determine how long it will take. This may include combinations of animation, path planning or other functionality. Examples include `drive_on_charger`, `set_lift_height`, etc." | Anki SDK |
| *boot loader* | A piece of software used to load and launch the application software. | |
| *C-like abstract data structure (CLAD)* | Anki's phrase for how they pack information into fields and values with a defined binary format. "Any data [passed] over the wire, [is] define[d with] enums, structures and messages in ".clad" files.. [with a] syntax [that] looks a lot like C structs. [A tool] auto-generate[s] Python, C++ and C# code for each of these structures, along with code to serialize and deserialize to efficiently packed byte streams of data."[55] (FlatBuffers are used for the same purpose, but were not available when CLAD was developped.) | |
| *capacitive touch* | A type of sensing where light contact, such as touch, is detected without requiring pressing a mechanism. | |
| *cascade* | Applies a series of fast to compute filters and classifiers to detect low-level features and identify things like faces. | |
| *certificate* | Vector generates an SSL certificate that can be used for the secure communications. | |
| *characteristic (Bluetooth LE)* | A key (or slot) that holds a value in the services key-value table. A characteristic is uniquely identified by its UUID. | |

[55] https://forums.anki.com/t/what-is-the-clad-tool/102/3

| | |
|---|---|
| *client token* | A string token provided by Vector that is passed with each SDK command. |
| *control* | Responsible for motors and forces to move where and how it is told to. (smooth arcs) |
| *D*-lite* | A path-finding algorithm |
| *device mapper verity (dm-verity)* | A feature of the Linux kernel that checks the boot and RAM file systems for alteration, using signed keys |
| *entitlement* | An entitlement is a family of features or resources that the program or owner is allowed to use. |
| *face detection* | The ability to realize that there is a face in the image, and where it is |
| *face recognition* | The ability to know the identity of a face seen. |
| *feature flags aka feature toggle* | A setting that enables and disables features, especially those still in development. This allows developing the code and integrating its structure before the module or function is completely ready. Otherwise it is very difficult to keep the different branches of development in sync and merge them when the feature is ready. |
| *field of view* | How wide of an area in the world that the camera can see |
| *firmware* | A type of software held (and usually executed from) in ROM or flash. It may have a (minimal) operating system, but often does not. |
| *flash* | A type of persistent (non-volatile) storage media. |
| *guidance* | Builds the desired path |
| *gyroscope* | A sensor that is used to measure how fast Vector is turning (the angular velocity) along its x, y, and z axes. |
| *Haar feature* | Facial features picked out using Haar wavelets |
| *Haar wavelet* | A fast, low-cost that can used to pick out (or recognize) simple features in an image. |
| *inner node* | A node in a tree data structure that does links to other nodes below it. Often it does not hold any other information. |
| *intent* | This is an internal code used to represent the command or question that corresponds to a phrase spoken by a person. |
| *Kalman filter* | Used to merge two or more noisy signals together to estimate a proper signal. |
| *leaf node* | A node in a tree data structure that does not link to any other nodes below it. It holds the information that was being looked up. |
| *navigation* | Knowing where it is in the map |
| *nonce* | An initially random number, incremented after each use. |
| *path planning* | Forms smooth arcs and line segments to move in around an environment to avoid collisions, blocked paths, and cliffs. |
| *pose* | The position and orientation of an object relative to a coordinate system |
| *power source* | Where the electric energy used to power Vector comes from. |
| *quad-tree* | A way of compressing a 2D map down into regions. |
| *rapidly-expanding random tree* | A path-finding algorithm |
| *recovery mode* | A separate, independent operating system that Vector can boot into for purposes of downloading software to replace a damaged partition. |

| | |
|---|---|
| *robot name* | Vector's robot name looks like "Vector-E5S6". It is "Vector-" followed by a 4 letters and numbers. |
| *session token* | A string token provided by the Anki servers that is passed to Vector to authenticate with him and create a *client token.* |
| *simultaneous localization and mapping* | A vision-based technique for building a map of the immediate world for purposes of positioning oneself within it and detecting relative movements. |
| *service (Bluetooth LE)* | A key-value table grouped together for a common purpose. A service is uniquely identified by its UUID. |
| *software* | Software is distinct from firmware in that is often loaded from external storage to be run in RAM, and is based on dynamic linking, allowing the use of other (replaceable) software elements. It does not access hardware directly; instead it employs sophisticated features of the operating system. |
| *syscon* | The name of the firmware program running on the base-board. |
| *text to speech* | A process of reading aloud a word, phrase, sentence, etc. |
| *trigger word* | aka wake word |
| *Trust Zone* | A security mode on ARM processor where privileged/special code is run. This includes access to encryption/decryption keys. |
| *universally unique identifier (UUID)* | A 128bit number that is unique. (effectively same as GUID) |
| *wake word* | The phrase ("Hey, Vector") used to activate Vector so that he will respond to spoken interaction. |

## 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.

***Table 201:*** *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<br>https://github.com/anki/avs-device-sdk https://developer.amazon.com/alexa-voice-service/sdk |
| *Amazon Simple Queue Service (SQS)* | Vector employs Amazon's SQS for its DAS functions. |
| *Amazon Simple Storage Service (S3)* | Vector's cloud interface uses Amazon's AWS go module to interact with Amazon's service:<br>https://docs.aws.amazon.com/sdk-for-go/api/service/s3/ https://docs.aws.amazon.com/AmazonS3/latest/API/API_Operations_Amazon_Simple_Storage_Service.html |
| *Amazon Web services* | used on the server https://aws.amazon.com/ |
| *android boot-loader* | Vector uses the Android Boot-loader; the code can be found in the earlier archive. |
| *ARM NN* | ARM's neural network support https://github.com/ARM-software/armnn |
| *AudioKinetic Wwise*[56] | Used to craft the sounds https://www.audiokinetic.com/products/wwise/ |
| *Backtrace.io* | A service that receives uploaded minidumps from applications in the field and provides tools to analyze them. https://backtrace.io |
| *clang* | A C/C++ compiler, part of the LLVM family https://clang.llvm.org |
| *bluez v5* | Bluetooth LE support http://www.bluez.org/ |
| *busybox* | The shell on the Anki Vector linux https://busybox.net |
| *chromium update* | ? |

[56] https://blog.audiokinetic.com/interactive-audio-brings-cozmo-to-life/

| | |
|---|---|
| *civetweb* | The embedded webserver that allows Mobile apps and the python SDK to communicate with Vector.<br>https://github.com/civetweb/civetweb |
| *connman* | Connection manager for WiFi<br>https://01.org/connman |
| *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 Vector's internal software. |
| *Google Breakpad* | Google Breakpad is used to generate tracebacks and mini-dump files of programs that crash. Results are sent to htttp://backtrace.io<br>https://chromium.googlesource.com/breakpad/breakpad |
| *Google FlatBuffers* | Google FlatBuffers is used to encode the animation data structures. "It is similar to protocol buffers, but the primary difference is that FlatBuffers does not need a parsing/unpacking step to a secondary representation before you can access data, often coupled with per-object memory allocation. Also, the code footprint of FlatBuffers is an order of magnitude smaller than protocol buffers"<br>[57]https://github.com/google/flatbuffers |
| *Google Protobuf* | Google's Protobuf interface-description language is 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.<br>https://developers.google.com/protocol-buffers |
| *Google RPC (gRPC)* | A "remote procedure call" standard, that allows mobile apps and the python SDK to communicate with Vector.<br>https://grpc.io/docs/quickstart/cpp/ |
| *hdr-histogram* | Unknown use<br>https://github.com/HdrHistogram/HdrHistogram |
| *libsodium* | Cryptography library suitable for the small packet size in Bluetooth LE connections. Used to encrypt the mobile applications Bluetooth LE connection with Vector.<br>https://github.com/jedisct1/libsodium |
| *linux, yocto*[58] | The family of linux distribution used for the Anki Vector<br>(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.<br>https://www.mpg123.de/index.shtml |
| *ogg vorbis* | Audio codec<br>https://xiph.org/vorbis |
| *Omron OKAO Vision* | Vector uses the Omron Okao Vision library for face recognition and tracking.<br>https://plus-sensing.omron.com/technology/position/index.html |
| *open CV* | Used for the first-level image processing – to locate faces, hands, and possibly accessory symbols. |

[57] https://nlp.gitbook.io/book/tensorflow/tensorflow-lite
[58] https://www.designnews.com/electronics-test/lessons-after-failure-anki-robotics/140103493460822

| | | |
|---|---|---|
| | https://opencv.org/ |
| *openssl* | used to validate the software update signature<br>https://www.openssl.org |
| *opkg* | Package manager, from yocto<br>https://git.yoctoproject.org/cgit/cgit.cgi/opkg/ |
| *Opus codec* | Audio codec; to encode speech sent to servers<br>http://opus-codec.org/ |
| *perl* | A programming language, on Victor<br>https://www.perl.org |
| *Pretty Fast FFT*<br>*pffft* | Julien Pommier's FFT implementation for single precision, 1D signals<br>https://bitbucket.org/jpommier/pffft |
| *Pryon, Inc* | The recognition for the Alexa keyword at least the file system includes the same model as distributed in AMAZONLITE<br>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.<br>https://www.python.org |
| *Qualcomm* | Qualcomm's device drivers, camera support and other kit are used. |
| *Segger ICD* | A high-end ARM compatible in-circuit debugging probe. Rumoured to have been used by Anki engineers, probably with the STM32F030<br>https://www.segger.com/products/debug-probes/j-link/ |
| *Sensory TrulyHandsFree* | Vectors recognition for "Hey Vector" and Alexa wake word is done by Sensory, Inc's TrulyHandsfree SDK 4.4.23 (c 2008)<br>https://www.sensory.com/products/technologies/trulyhandsfree/<br>https://en.wikipedia.org/wiki/Sensory,_Inc. |
| *Signal Essence* | Designed the microphone array, and the low-level signal processing of audio input.<br>https://signalessence.com/ |
| *Sound Hound, inc* | Vector's Q&A "knowledge graph" is done by Sound Hound<br>https://blog.soundhound.com/hey-vector-i-have-a-question-3c174ef226fb |
| *SQLite* | This is needed by Alexa; other uses are unknown<br>https://www.sqlite.org/index.html |
| *systemd* | Used by Vector to launch the internal services<br>https://www.freedesktop.org/software/systemd/ |
| *tensor flow lite (TFLite)* | TensorFlow lite is used to recognize hands, the desk surface, and was intended to support recognizing pets and common objects.<br>https://www.tensorflow.org/lite/microcontrollers/get_started |

# 108. REFERENCES & RESOURCES

Several of the tools have licenses requiring Anki to post that the tools was listed and/or to post their versions of the tools, and their modification. The following archives of *some* of the open source tools are listed in the "acknowledgements" section of the mobile application:[59]

https://anki-vic-pubfiles.anki.com/license/prod/1.0.0/licences/OStarball.v160.tgz
https://anki-vic-pubfiles.anki.com/license/prod/1.0.0/licences/engineTarball.v160.tgz

[59] You can only read the acknowledgements in the mobile application if you are connected to a robot. The mobile app is not great.

Notes: Other open source tools used by Anki were used without Anki posting their version (or modifications).

## Appendix C

# Alexa modules

This Appendix outlines the modules used by the Alexa client built into Vector (using the Alexa Client SDK). Alexa's modules connect together like so:

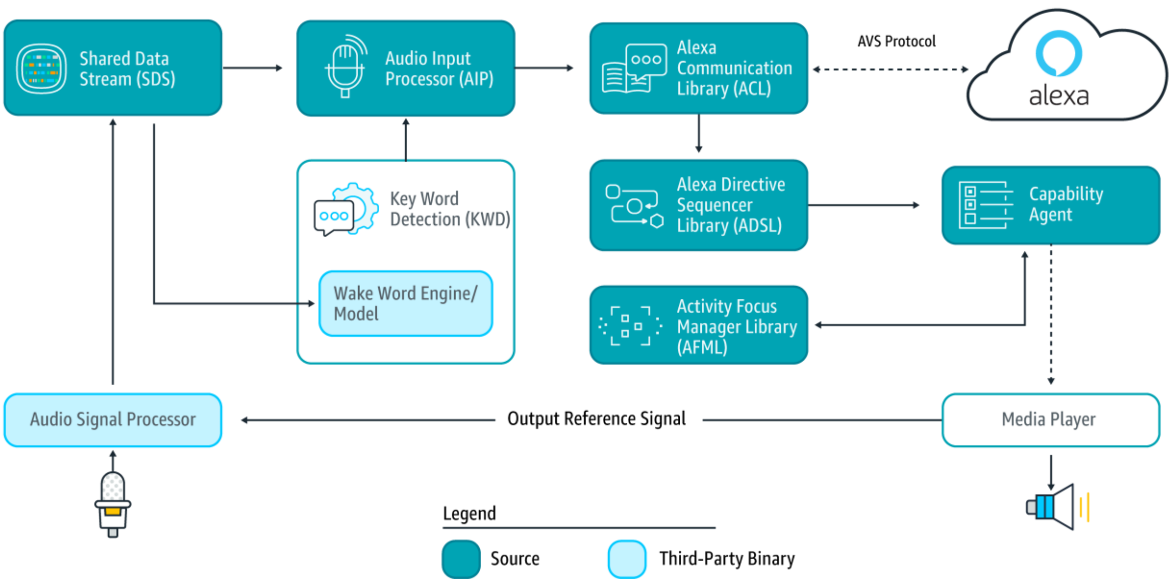



***Figure 61:** Alexa's function blocks* (image courtesy Amazon)

Alexa's modules include:

***Table 202:** Alexa files*

| Library | Description & Notes |
|---|---|
| libACL.so | Alexa Communication Library. "Serves as the main communications channel between the device and the Alexa Voice Service." |
| libAIP.so | Audio Input Processor. "Handles the audio input to Alexa Voice Service from on-device microphones, remote microphones and other audio input sources." |
| libADSL.so | Alexa Directive Sequencer Library (Directive Router, Processor, Sequencer; Message Interpreter). |
| libAFML.so | Activity Focus Manager Library, including Audio Activity Tracker, Visual Activity tracker. "Prioritizes the channel inputs and outputs as specified by the AVS Interaction Model" |
| libAlerts.so | Alexa alert scheduler; "The interface for setting, stopping, and deleting timers and alarms." |
| libAudioPlayer.so | Alexa's audio player. "The interface for managing |

| | |
|---|---|
| | and controlling audio playback.” |
| libAudioResources.so | Alexa’s audio resources, including calls |
| libAVSCommon.so | Alexa’s voice service support |
| libAVSSystem.so | Alexa’s voice service support |
| libCapabilitiesDelegate.so | Alexa capabilities. “Handles Alexa-driven interactions; specifically, directives and events. Each capability agent corresponds to a specific interface exposed by the AVS API.” |
| libCBLAuthDelegate.so | Alexa Authorization |
| libCertifiedSender.so | Alexa certified sender |
| libContextManager.so | Alexa’s context manager |
| libESP.so | Alexa ESP, Dummy ESP |
| libInteractionModel.so | “This interface allows a client to support complex interactions initiated by Alexa, such as Alexa Routines.” |
| libNotifications.so | Alexa Notifications. “The interface for displaying notifications indicators.” Uses SQLite |
| libPlaybackController.so | “The interface for navigating a playback queue via GUI or buttons.” |
| libPlaylistParser.so | Alexa playlist |
| libRegistrationManager.so | Alexa’s registration manager |
| libSettings.so | Alexa’s settings & preferences module |
| libSpeakerManager.so | |
| libSpeechSynthesizer.so | “The interface for Alexa speech output.” |

*Note: quotes from Amazon Alexa Voice Services SDK documentation*

APPENDIX D

# Fault and status codes

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

*Table 203: The system fault codes*

| Code | Meaning |
|---|---|
| 1..10 | Systemd failed…? |
| 200… | Software update status code, see table below |
| 700-702 | Internal sensor out of range or failed. These require vic-robot to tell the base-board to power the system off. |
| 703-705 | Internal sensor out of range or failed. |
| 800 | Vic-anim was unable to start or crashed. |
| 801 | ? |
| 898 | "general hardware disconnect" Perhaps vic-robot is unable to communicate with the base-board? |
| 899 | ? |
| 913 | Vic-switchboard was unable to start or crashed |
| 914 | Vic-engine was unable to start or crashed |
| 915 | Vic-engine stopped responding. |
| 916 | Vic-robot was unable to start or crashed |
| 917 | Vic-anim stopped responding |
| 920 | Vic-gateway-cert was unable to generate a x509 certificate for vic-gateway |
| 921 | Vic-gateway was unable to start or crashed |
| 923 | Vic-cloud was unable to start or crashed |
| 980-981 | "These codes indicate issues with the camera. These issues are typically caused by mm-anki-camera hanging when we try to stop the camera stream on vic-engine stop. We have to manually kill it and start it again." |

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

*Table 204: OTA update-engine status codes*

| Status | Meaning |
|---|---|
| 200 | The TAR contents did not follow the expected order. |
| 201 | Unhandled section format for expansion, or<br>The manifest version is not supported, or<br>The OTA has the wrong number of images for the type, or<br>The OTA is missing a BOOT or SYSTEM image, or<br>The manifest configuration is not understood |
| 202 | Could not mark target, a, or b slot unbootable, or<br>Could not set target slot as active |
| 203 | Unable to construct automatic update URL, or<br>The URL could not be opened |

| | |
|---|---|
| 204 | The file wasn’t a valid TAR file, or is corrupt |
| 205 | The compression scheme is not supported, or<br>Decompression failed, the file may be corrupt |
| 207 | Delta payload error |
| 208 | Couldn't sync OS images to disk, or<br>Disk error while transferring OTA file. |
| 209 | The manifest failed signature validation; or the `aboot`, `boot` image, `system` image, or `delta.bin` hash doesn't match signed manifest |
| 210 | The encryption scheme is not supported. |
| 211 | Vector’s current version doesn’t match the baseline for a delta update. |
| 212 | The decompression engine had an unexpected, undefined error. |
| 213 | QSN doesn't match manifest |
| 214 | There is a mismatch: development Vectors can’t install release OTA software, and release Vectors can’t install development OTA software. |
| 215 | OTA transfer failed, due to timeout. |
| 216 | OS version name in the update file doesn’t follow an acceptable pattern, or it is not allowed to upgrade or downgrade from the current version to the new version. |
| 219 | Other unexpected, undefined error while transferring OTA file. |

## APPENDIX E

# File system

This Appendix describes the file systems on Vector's flash. As the Vector uses the Android bootloader, it reuses – or at least reserves – many of the Android partitions[60] 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 205:** The file system mount table*

| Mount point | Partition name | Description & Notes |
|---|---|---|
| / | BOOT_A | The primary linux kernel and initramfs |
| /data[61] | 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<br>/run<br>/var/volatile<br>/dev/sm | | Internal temporary file systems; holds temporary files, interprocess communication |

The partition table[62] found on the Vector:

***Table 206:** The partition table*

| Partition name | Size | Description & Notes |
|---|---|---|
| ABOOT<br>ABOOTBAK | 1 MB<br>1 MB | The primary and backup Android boot loader, which may load the kernel, recovery, or fastboot. This is in the format of a signed, statically linked ELF binary. |
| BOOT_A<br>BOOT_B | 32 MB<br>32 MB | These are the primary and backup linux kernel and initramfs. Updates modify the non-active partition, and then swap which one is active. |
| CONFIG | 512 KB | This partition is not employed by Vector. It is zero'd out. |
| DDR | 32 KB | Configuration of the DDR RAM. |
| DEVINFO | 1 MB | This partition is not read by Vector. It is zero'd out.<br>In typical aboot implementations this partition is used to hold "device |

[60] https://forum.xda-developers.com/android/general/info-android-device-partitions-basic-t3586565
[61] This is mounted by "mount-data.service" The file has a lot of information on how it unbricks
[62] Much information from: https://source.android.com/devices/bootloader/partitions-images

| | | |
|---|---|---|
| | | information including: is_unlocked (aboot), is_tampered, is_verified, charger_screen_enabled, display_panel, bootloader_version, radio_version etc. Contents of this partition are displayed by "fastboot oem device-info" command in human readable format. Before loading boot.img or recovery.img, [the] boot loader verifies the locked state from this partition." Vector's aboot will write to this partition to indicate tampering when it finds that the boot image does not pass integrity checks. |
| EMR | 16 MB | This is Vectors "Electronic Medical Record." It holds Vector's Model, Serial Number, and such. It is a binary data structure, rather than a file system. |
| 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 | This is "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<br>MODEMST2 | 1.5MB<br>1.5MB | A FAT file-system holding executables and binary "blobs" for the WiFi/Bluetooth radio firmware. Several are signed by Anki. Includes a lot of test code, probably for emissions testing. |
| OEM | 16MB | A modifiable ext2/4 file system that holds the logs, robot name, some calibration info, and SDK TLS certificates. |
| PAD | 1MB | "related to OEM" |
| PERSIST | 64MB | This partition is not employed by Vector. It is zero'd out. |
| RECOVERY | 32 MB | An alternate partition holding kernel and initial RAM filesystem that allows the system 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. This partition holds v0.90 of the Anki software. |
| RPM<br>RPMBAK | 512KB<br>512KB | The primary and backup partitions for resource and power management. This is in the format of a signed, statically linked ELF binary. |
| SBL1<br>SBL1BAK | 512KB<br>512KB | The primary and back up partitions for the secondary boot-loader. Responsible for loading aboot; has an "Emergency" download (EDL) mode using Qualcomm's Sahara protocol. This is in the format of a signed, statically linked ELF binary. |
| 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<br>SYSTEM_B | 896MB<br>896MB | The primary and backup system applications and libraries with application specific code. Updates modify the non-active partition, and then swap which one is active. |
| SWITCHBOARD | 16 MB | This is a modifiable data area used by Vic-switchboard to hold persistent communication tokens. This appears to be a binary data structure, rather than a file system. |
| TZ | 768KB | The primary and backup TrustZone. This is in the format of a signed, statically |

| | | |
|---|---|---|
| TZBAK | 768KB | linked ELF binary. This code is executed with special privileges to allow encrypting and decrypting key-value pairs without any other modules (or debuggers) having access to the secrets. |
| 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 following files are employed in the Vector binaries and scripts:

***Table 207:*** *Files*

| File | Description |
|---|---|
| */anki/etc/revision* | Contains the robot revision number |
| */anki/etc/version* | Contains the robot version number |
| */data/data/com.anki.victor* | |
| */data/data/com.anki.victor/cache/crashDumps* | Holds the crash dump files |
| */data/data/com.anki.victor/cache/outgoing* | |
| */data/data/com.anki.victor/cache/vic-logmgr* | A folder used to hold the log files while constructing the compressed archive file that will be uploaded. |
| */data/diagnostics/* | Used to hold the diagnostic logs as the archive is constructed and compressed. |
| */data/etc* | |
| */data/etc/localtime* | The time zone |
| */data/fault-reports* | |
| */data/lib/connman/* | The contents of `/var/lib/connman` are copied here |
| */data/maintenance_reboot* | This is set when the system has rebooted for maintenance reasons (e.g. updates) |
| */data/misc/bluetooth* | A folder to hold communication structures for the Bluetooth LE stack. |
| */data/misc/bluetooth/abtd.socket* | The IPC socket interface to Anki's Bluetooth LE service |
| */data/misc/bluetooth/btprop* | The IPC socket interface to BlueZ Bluetooth LE service. |
| */data/misc/camera* | |
| */data/panics* | |
| */data/run/connamn* | |
| */data/data/com.anki.victor/persistent/switchboard/sessions* | Used by Vic-switchboard to hold persistent session information, e.g. tokens |
| */data/unbrick* | |
| */data/usb* | |
| */data/vic-gateway* | |
| */dev/block/bootdevice/by-name/emr* | File system access to the manufacturing records, including serial number |
| */dev/block/bootdevice/by-name/switchboard* | File system access to switchboards persistent data. |
| */dev/rampost_error* | The status of the rampost checks of the baseboard. |
| */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 |
| */etc/os-version* | Contains the OS (linux) version string. |
| */etc/os-version-rev* | Contains the OS (linux) revision string. |
| */factory/cloud/something.pem* | |
| */proc/sys/kernel/random/boot_id* | A random identifier, created each boot |
| */sys/devices/system/cpu/possible*[63] */sys/devices/system/cpu/present* | The number of CPUs and whether they can be used. |
| */run/after_maintenance_reboot* | This is set to indicate to Vectors services that the system was rebooted for maintenance reasons, and they should take appropriate action. This will be set, on boot, if `/data/maintenance_reboot` had been set. |
| */run/anki-crash-log* | |
| */run/das_allow_upload* | If this exists, the crash log files can be uploaded to the backtrace.io servers; if it does not exist, the files are not uploaded. This file probably always exists, but was intended to be a user settable feature. |
| */run/fake-hwclock-cmd*[64] | Sets the fake time to the time file (Vector doesn't have a clock) |
| */run/fault_code* | This is set to the fault code (see Appendix C) if a program is unable to carry out a significant task, or crashes. The fault display program may present this code on the LCD display. |
| */run/fault_code.pending* | The next fault code in queue to be handled |
| */run/fault_code.showing* | The fault code being displayed |
| */run/fault_restart_count* | This is incremented with each restart, and cleared by a reboot. |
| */run/fault_restart_uptime* | |
| */tmp/data_cleared* | |
| */tmp/vision/neural_nets* | |

Key named device files employed in Vector binaries:

***Table 208:** Named device and control files*

| File | Description |
|---|---|
| */dev/fb0* | The display framebuffer |
| */dev/spidev0.0* | The SPI channel to communicate with the IMU |
| */dev/spidev1.0* | The SPI channel to communicate with the LCD |
| */dev/ttyHS0* | Serial connection with the base-board |
| */dev/ttyHSL0* | Console log |
| */sys/class/android_usb/android0/iSerial* | Set to Vector's serial number |
| */sys/class/gpio/gpio83* | Used to control the camera power |
| */sys/class/leds/face-backlight-left/brightness* | LCD left backlight control |
| */sys/class/leds/face-backlight-right/brightness* | LCD right backlight control |
| */sys/devices/platform/soc/1000000.pinctrl/gpio/gpiochip0/base* | LCD backlight enable (left or right?) GPIO config |

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

| | |
|---|---|
| */sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq* | The maximum frequency that the CPU can run at. Initially set to 533MHz |
| */sys/kernel/debug/msm_otg/bus_voting* | Disabled to prevent the USB from pinning RAM to 400MHz. |
| */sys/kernel/debug/rpm_send_msg/message* | Used to control the RAM controller. The RAM is set to a maximum of 400MHz. |
| */sys/devices/soc/1000000.pinctrl/gpio/gpiochip0/base* | LCD backlight enable (left or right?) GPIO config |
| */sys/devices/soc.0/1000000.pinctrl/gpio/gpiochip911/base* | LCD backlight enable (left or right?) GPIO config |
| */sys/module/spidev/parameters/bufsiz* | The buffer size for SPI transfers. This is set to the size of the LCD frame (184 pixels × 96 pixels × 2 bytes/pixel). |

APPENDIX F

# 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.

## 109. CUBE SERVICES

times and other feature parameters:

| Service | UUID[65] | Description & Notes |
|---|---|---|
| *Device Info Service*[66] | $180A_{16}$ | Provides device and unit specific info –it's manufacturer, model number, hardware and firmware versions |
| *Generic Access Profile*[67] | $1800_{16}$ | The device name, and preferred connection parameters |
| *Generic Attribute Transport*[68] | $1801_{16}$ | 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 209:** 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 210:** The Cube's Device info settings*

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

## 109.1. CUBE'S ACCELEROMETER SERVICE

Values are little-endian, except where otherwise stated.

*Table 211: Cube's accelerometer service characteristics*

| UUID | Access | Size | Notes |
|---|---|---|---|
| 0EA75290-6759-A58D-7948-598C4E02D94A$_{16}$ | *Write* | unknown | |
| 450AA175-8D85-16A6-9148-D50E2EB7B79E$_{16}$ | Read | The date and time of manufacture (?) | |
| | | char[] | *A date and time string* |
| 43EF14AF-5FB1-7B81-3647-2A9477824CAB$_{16}$ | Read, Notify, Indicate | Reads the battery and accelerometer | |
| | | 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 |

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

# 110. VECTOR SERVICES SERVICE

times and other feature parameters:

*Table 212: Vector's Bluetooth LE services*

| Service | UUID[69] | Description & Notes |
|---|---|---|
| *Generic Access Profile* | 1800$_{16}$ | The device name, and preferred connection parameters |
| *Generic Attribute Transport* | 1801$_{16}$ | Provides access to the services. |
| *Vector's Serial Service* | FEE3$_{16}$ | The service with which we can talk to Vector. |

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

*Table 213: The Vector's Device info settings*

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

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

## 110.1. VECTOR'S SERIAL SERVICE

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

***Table 214:*** *Vector's serial service characteristics*

APPENDIX G

# Servers & Data Schema

This Appendix describes the servers that Vector contacts[70]

*Table 215: The servers that Vector contacts.*

| 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.[71] |
| https://anki.sp.backtrace.io:6098/post?format=minidump&token=6fd2bd053e8dd542ee97c05903b1ea068f090d37c7f6bbfa873c5f3b9c40b1d9 | Vector posts crashes (linux minidumps) to this server. This is hard coded in anki-crashuploader |
| 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 | |

The mobile application contacts the following servers:

*Table 216: The servers that the mobile application contacts.*

| Server | Description & Notes |
|---|---|
| https://locations.api.anki.com/1/locations | This is used to provide a list of locations to the mobile application that the Chipper servers will recognize. Without this, you cannot change Vector's location in the mobile application |

The Alexa modules contact the following servers:

*Table 217: The Amazon Alexa Voice Service servers that Vector contacts.*

| Server | Description & Notes |
|---|---|
| https://api.amazon.com/auth/O2/ | Used to authenticate the account for the Alexa |

70 Todo: sync up with info at: https://github.com/anki-community/vector-archive
71 Project Victor had a write up, reference that.

| | |
|---|---|
| | device. |
| `https://avs-alexa-na.amazon.com` | The Alexa Voice Service that accepts the spoken audio and returns a rich intent. Amazon changed preferred URLs on 2019 May 22, and this is considered legacy.[72] |

[72] https://developer.amazon.com/docs/alexa-voice-service/api-overview.html

## APPENDIX H

# Features

The following is the set of features and whether they are enabled:

*Table 218: The features*

| Feature | enabled | Description & Notes |
|---|---|---|
| *ActiveIntentFeedback* | true | |
| *Alexa* | true | The ability to use Alexa |
| *Alexa_AU* | true | The ability to use Alexa, localized for Australia |
| *Alexa_UK* | true | The ability to use Alexa, localized for the UK |
| *AttentionTransfer* | false | |
| *CubeSpinner* | false | |
| *Dancing* | true | The ability for Vector to dance to music. |
| *Exploring* | true | The ability for Vector to explore his area |
| *EyeColorVC* | true | The ability to set Vector's eye color through a voice command |
| *FetchCube* | true | The ability for Vector to fetch his cube |
| *FindCube* | true | The ability for Vector to find his cube |
| *GazeDirection* | false | |
| *GreetAfterLongTime* | true | |
| *HandDetection* | true | The ability for Vector to spot hands |
| *HeldInPalm* | true | |
| *HowOldAreYou* | true | The ability for Vector to track how long it has been since he was activated (his age) and use that info to respond to the question "How old are you?" |
| *Invalid* | false | |
| *Keepaway* | true | |
| *KnowledgeGraph* | true | The ability for Vector to answer a question when asked "Hey Vector, I have a question…" |
| *Laser* | false | |
| *Messaging* | false | |
| *MoveCube* | true | |
| *PopAWheelie* | true | The ability for Vector pop a wheelie using his cube |
| *PRDemo* | false | |

| | | |
|---|---|---|
| *ReactToHeldCube* | true | |
| *ReactToIllumination* | true | |
| *RollCube* | true | The ability for Vector to drive up and roll his cube |
| *StayOnChargerUntilCharged* | true | |
| *TestFeature* | false | |
| *Volume* | true | The ability to set Vector’s volume by voice command. |

APPENDIX I

# Phrases and their Intent

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

**Table 219:** *The "Hey Vector" phrases*

| Intent | Enumeration | Phrase |
|---|---|---|
| *movement_backward* | 23 | Back up |
| *imperative_scold* | 18 | Bad robot |
| *imperative_quiet* | | Be quiet |
| *global_stop* | 3 | Cancel the timer |
| | | Change/set your eye color to [blue, green, lime, orange, purple, sapphire, teal, yellow]. |
| *check_timer* | 1 | Check the timer |
| *imperative_come* | 10 | Come here |
| *imperative_dance* | 11 | Dance. |
| *play_popawheelie* | 34 | Do a wheelstand |
| *imperative_fetchcube* | 12 | Fetch your cube |
| *imperative_findcube* | 13 | Find your cube |
| *play_fistbump* | 32 | Fist Bump |
| *play_fistbump* | 32 | Give me a Fist Bump |
| *movement_backward* | 23 | Go backward |
| *explore_start* | 2 | Go explore |
| *movement_forward* | 22 | Go forward. |
| *movement_turnleft* | 24 | Go left |
| *movement_turnright* | 25 | Go right |
| *system_sleep* | | Go to sleep |
| *system_charger* | | Go to your charger |
| | | Good afternoon |
| *greeting_goodbye* | 4 | Goodbye |

| | | |
|---|---|---|
| | | Good evening |
| *greeting_goodnight* | | Good night |
| *greeting_goodmorning* | 5 | Good morning |
| *imperative_praise* | 16 | Good robot |
| *seasonal_happyholidays* | 36 | Happy Holidays |
| *seasonal_happynewyear* | 37 | Happy New Year |
| *greeting_hello* | 6 | Hello |
| | | He's behind you |
| *character_age* | 0 | How old are you |
| *imperative_abuse* | 7 | I hate you. |
| *knowledge_question* | 27 | I have a question … |
| *imperative_love* | 15 | I love you. |
| *imperative_apology* | 9 | I'm sorry. |
| *play_blackjack* | 31 | Let's play Blackjack |
| | | Listen to music |
| *imperative_lookatme* | 14 | Look at me |
| | | Look behind you |
| | | My name is [Your Name] |
| *imperative_negative* | 17 | No |
| *play_pickupcube* | 33 | Pick up your cube. |
| *play_anygame* | 29 | Play a game |
| *play_anytrick* | 30 | Play a trick |
| *play_blackjack* | 31 | Play Blackjack |
| *play_popawheelie* | 34 | Pop a wheelie. |
| *play_rollcube* | 35 | Roll your Cube |
| *imperative_quiet* | | Quiet down |
| | | Run |
| *set_timer* | 38 | Set a timer for [length of time] |
| *imperative_shutup* | | Shut up |
| *explore_start* | 2 | Start Exploring |
| | | Stop Exploring |
| *global_stop* | 3 | Stop the timer |
| *take_a_photo* | 40 | Take a picture of [me/us] |
| *take_a_photo* | 40 | Take a picture |
| *take_a_photo* | 40 | Take a selfie |

| | | |
|---|---|---|
| *movement_turnaround* | 26 | Turn around |
| *movement_turnleft* | 24 | Turn left |
| *{same as be quiet }* | | Turn off |
| *movement_turnright* | 25 | Turn right |
| *imperative_volumelevel* | 19 | Volume [number]. |
| *imperative_volumedown* | 21 | Volume down |
| *imperative_volumeup* | 20 | Volume up. |
| | | Volume maximum |
| *names_ask* | 28 | What's my name? |
| *weather_response* | 41 | What's the weather in [City Name]? |
| *weather_response* | 41 | What's the weather report? |
| *show_clock* | 39 | What time is it? |
| *blackjack_hit* | | |
| *blackjack_playagain* | | |
| *blackjack_stand* | | |
| *global_delete* | | |
| *imperative_lookoverthere* | | |
| *knowledge_response* | | |
| *knowledge_unknown* | | |
| *meet_victor* | | |
| *message_playback* | | |
| *message_record* | | |
| *silence* | | |
| *status_feeling* | | |
| *imperative_affirmative* | 8 | Yes |

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

Questions

***Table 220:*** *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 J

# DAS Tracked Events and Statistics

This Appendix captures the events and statistics that are posted to Anki's the diagnostics / analytics services (see Chapter 20)

## 111. DAS TRACKED EVENTS AND STATISTICS

### 111.1. BASIC INFORMATION

#### 111.1.1 Version Information

The following are version-information events that are posted to the diagnostic logger:

| Event | Description & Notes |
|---|---|
| hal.body_version | |
| robot.boot_info | |
| robot.cpu_info | |

***Table 221:** Version info, posted to DAS*

#### 111.1.2 Settings and Preferences Information

The following are settings and preference related events that are posted to the diagnostic logger:

| Event | Description & Notes |
|---|---|
| das.allow_upload | |
| engine.language_locale | |
| robot.locale | |
| robot.timezone | |

***Table 222:** Start up information, posted to DAS*

#### 111.1.3 Start-up Information not described elsewhere

The following are start-up events that are posted to the diagnostic logger:

| Event | Description & Notes |
|---|---|
| ntp.timesync | |

***Table 223:** Settings and preferences, posted to DAS*

| | |
|---|---|
| profile_id.start | |
| rampost.lcd_check | |
| rampost.rampost.exit | The rampost has completed and exited. |
| random_generator.seed | |
| robot.engine_ready | |
| switchboard.hello | |
| vic.cloud.hello.world | |

*Note: other startup events are covered elsewhere with their functional groups.*

## 111.2. POWER MANAGEMENT EVENTS AND STATISTICS

The power management posts the following set of related events:

| Event | Description & Notes |
|---|---|
| hal.active_power_mode | |
| PowerModeManager.DisableActiveMode | |
| PowerModeManger.EnableActiveMode | |
| robot.power_on | |

***Table 224:*** *Power management events, posted to DAS*

### 111.2.1 Battery Statistics and Events

The battery management posts the following battery related events and state information:

| Event | Description & Notes |
|---|---|
| battery.battery_level_changed | This is set when the battery level has changed from event posting. |
| battery.encoder_power_stats | ?? Strange name. Is this the voltage seen on the charger input and charging duration stats? |
| battery.fully_charged_voltage | The battery voltage seen when the charger reported the battery to be fully charged. |
| battery.voltage_reset | |
| battery.voltage_stats | Information about the range of battery voltages that have been observed; e.g. min/max, average, etc. |
| rampost.battery_flags | |
| rampost.battery_level | |

***Table 225:*** *Battery level events and statistics*

### 111.2.2 Charger Statistics and Events

The charging function of the battery management system posts the following events and state information:

*Table 226: Charger statistics and events, posted to DAS*

| Event | Description & Notes |
|---|---|
| battery.cooldown | Indicates that Vector is or needed to pause charging and activity to let the battery cool down. |
| battery.is_charging_changed | This is set when the state of charging has changed from event posting. |
| battery.on_charger_changed | |
| battery.saturation_charging | |
| battery.temp_crossed_threshold | |
| battery.temperature_stats | Information about the range of battery temperatures that have been observed; e.g. min/max, average, etc. |
| rampost.battery_temperature | |

## 111.3. MOTOR AND IMU STATISTICS AND EVENTS

The motor controllers post the following events and statistics:

*Table 227: Motor events*

| Event | Description & Notes |
|---|---|
| calibrate_motors | |
| head_motor_calibrated | |
| head_motor_uncalibrated | |
| imu_filter.gyro_calibrated | |
| lift_motor_calibrated | |
| lift_motor_uncalibrated | |

## 111.4. COMMUNICATION RELATED EVENTS POSTED TO DAS

### 111.4.1 Base-Board / Spine Related Events Posted to DAS

The communication with the based-board controller posts the following events:

*Table 228: Base-board / spine related DAS events*

| Event | Description & Notes |
|---|---|
| rampost.spine.configure_serial_port | The rampost program was successful in configuring the serial port to communicate with the base-board. |
| rampost.spine.open_serial | The rampost was able to open the serial port to communicate with the base-board. |
| rampost.spine.select_timeout | There was a timeout in communicating with rampost the base-baoard. |

| | |
|---|---|
| touch_sensor.activate_charger_mode_check | |
| touch_sensor.charger_mode_check.no_baseline_change | |
| ttyHSL0.service | |

*Note: see the updates section for events related to updating the base-board firmware*

### 111.4.2 Accessory-Related Events Posted to DAS

The communication with the mobile application and SDK posts the following events:

***Table 229:** Accessory cube related DAS events*

| Event | Description & Notes |
|---|---|
| cube.battery_voltage | The cube's battery voltage |
| cube.connected | Vector was able to connect to his accessory companion cube. |
| cube.connection_failed | Vector was unable to connect to his accessory companion cube. |
| cube.disconnected | Vector lost the connection with his accessory companion cube. |
| cube.firmware_mismatch | Vector is unable to use his accessory companion cube as is, since the firmware version is not compatible. |
| cube.unexpected_connect_disconnect | |

*Note: see the updates section for events related to updating the cube firmware*

### 111.4.3 Mobile-App / SDK Related Events Posted to DAS

The communication with the mobile application and SDK posts the following events:

***Table 230:** Mobile application / SDK related DAS events*

| Event | Description & Notes |
|---|---|
| ble.connection | |
| ble_conn_id.stop | |
| ble.disconnection | |
| wifi.initial_state | |

## 111.5. UPDATE-RELATED EVENTS POSTED TO DAS

The following are events are posted by the update subsystem:

***Table 231:** Update events*

| Event | Description & Notes |
|---|---|
| cube.firmware_flash_success | |
| rampost.dfu.desired_version | |
| rampost.dfu.installed_version | |

| | |
|---|---|
| rampost.dfu.open_file | |
| rampost.dfu.request_version | |
| robot.ota_download_end | On success the parameters include the new version; on failure the parameters include the version identifier, error code, and some explanatory text. |
| robot.ota_download_stalled | |
| robot.ota_download_start | |

## 111.6. BEHAVIOUR, FEATURE, MOOD, AND ENGINE RELATED EVENTS POSTED TO DAS

The engine/animation controller post the following behavior-related events:

***Table 232:** Behaviour, feature, mood and engine related DAS events*

| Event | Description & Notes |
|---|---|
| action.play_animation | The specified animation will be played |
| behavior.feature.end | |
| behavior.feature.pre_start | The behaviour for specified feature will begin. |
| behavior.feature.start | The behaviour for specified feature are begun. |
| behavior.hlai.change | There was a change in the high-level AI state. Some possible supplemental parameters include "ObservingOnCharger" |
| dttb.coord_activated | The "dance to the beat" feature has been activated. |
| dttb.coord_no_beat | |
| engine.state | |
| mood.event | |
| mood.simple_mood_transition | |
| robot.object_located | |
| robot.reacted_to_sound | |

## 111.7. ROBOT LIFETIME STATISTICS & EVENTS

Vector tracks the following behavior and events

***Table 233:** The robot lifetime stats schema*

| Key | units | Description & Notes |
|---|---|---|
| Alive.seconds | seconds | Vector's 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 | | |
| Odom.LWheel | | The left wheel odometer |
| Odom.Rwheel | | The right wheel odometer |
| Odom.Body | | |

| | | |
|---|---|---|
| Pet.ms | ms | The number of milliseconds petted? |

## APPENDIX K

# 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."[73]

## 111.8. 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[74]

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

## 111.9. 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

[73] https://news.ycombinator.com/item?id=17755596
[74] 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